GPH-3300

Analog Output Board Driver Software for Linux/RTLinux

Help for Linux

www.interface.co.jp

Contents

Chapter 1 Introduction

1.1 Summary

The GPH-3300 software controls Interface analog output boards from your application running on Linux or RTLinux. Application software should link controls the analog output boards through the provided application programming interface (API). This document includes the information for using the GPH-3300 on Linux.

1.2 Features

- The GPH-3300 supports up to 2^{30} data. The maximum number of actual data that you can handle at a time depends on the amount of memory installed on your computer.

- Analog output can synchronously start or stop with triggers. The trigger delay function enables to delay the start or stop of analog output.

- Output range selection and offset/gain calibration are programmable by software. (Some boards don't support them.)

- The GPH-3300 supports every Interface analog output board.

- The GPH-3300 supports the parallel analog output update, so you can simultaneously analog output on two or more boards.

- The GPH-3300 provides the calibration program.

- The GPH-3300 provides useful sample program, they help you to develop the application programs.

- The analog output data can be saved into the disk, and the saved data can be used with various application or programs.

- The GPH-3300 supports data conversion from binary to physical value and vice versa.

- The noise can be removed by using the averaging.

Chapter 2 Product Specifications

2.1 Operating Environments

The following table shows operating environments for the GPH-3300.

Interface Single Board Computer	Contact us.
Interface Mother Board	Contact us.
Computer	Intel Architecture-32 (IBM PC/AT Compatibles)
Driver Type	Character driver
Loading Method	Loadable module
Major Number	Automatic assignment
Source Code Open Policy	Driver module: partially open Library source code: closed Common module: open
Build Support	Makefile provided
Help File	PDF format
	Text format

2.2 Target Boards

- PCI expansion boards (PCI series)

Major Data Transfer Mode	Model			
Programmed I/O	PCI-3310	PCI-3325	PCI-3329	PCI-3336
	PCI-3338	PCI-3340	PCI-3341A	PCI-3342A
	PCI-3343A	PCI-3345A	PCI-3346A	PCI-3347
	PCI-3521 (DA)	PCI-3522A (DA)	PCI-3523A (DA)	PCI-3525 (DA)
Memory	PCI-3305	PCI-3335	PCI-3337	
Bus master	PCI-3174 (DA)	PCI-3175 (DA)	PCI-3176 (DA)	

- PCI expansion boards (PAZ series)

Major Data Transfer Mode	Model			
Programmed I/O	PAZ-3310	PAZ-3325	PAZ-3329	PAZ-3336
	PAZ-3338	PAZ-3340	PAZ-3521 (DA)	
Memory	PAZ-3305			
Bus master	PAZ-3174 (DA)	PAZ-3176 (DA)		

- CompactPCI expansion boards

Major Data Transfer Mode	Model			
Programmed I/O	CTP-3174 (DA)	CTP-3175 (DA)	CTP-3182 (DA)	CTP-3325
	CTP-3329	CTP-3338	CTP-3340A	CTP-3340B
	CTP-3340C	CTP-3340D	CTP-3342	CTP-3343
	CTP-3346	CTP-3347	CTP-3348	CTP-3349
	CTP-3350	CTP-3351	CTP-3521 (DA)	CTP-3522 (DA)
	CTP-3523 (DA)			

2.3 Functional Specifications

<table>
<tr><th>Function</th><th>Description/Specification</th></tr>
<tr><td>Number of boards</td><td>(255 boards (max.) Up to 16 devices for the same type boards.</td></tr>
<tr><td>Number of channels</td><td>Up to the sum of channels of the boards installed on the system.</td></tr>
<tr><td rowspan="4">Data transfer mode</td><td>Programmed I/O</td></tr>
<tr><td>Memory</td></tr>
<tr><td>FIFO</td></tr>
<tr><td>Bus master</td></tr>
<tr><td>Output update rates</td><td>
<table>
<tr><th>Data Transfer Mode</th><th>Output Updata Rates</th></tr>
<tr><td>Programmed I/O</td><td>0.01 Hz to 80 kHz</td></tr>
<tr><td>FIFO</td><td>122 Hz to 100 kHz</td></tr>
<tr><td>Memory</td><td>0.01 Hz to 200 kHz</td></tr>
<tr><td>Memory</td><td>0.01 Hz to 5 MHz
(only for the PCI/PAZ-3305)</td></tr>
</table>
</td></tr>
<tr><td rowspan="2">Trigger capabilities</td><td>External trigger</td></tr>
<tr><td>External trigger with mask using general purpose digital input pins</td></tr>
<tr><td>Trigger timing</td><td>Analog output start-trigger and stop-trigger are available.</td></tr>
<tr><td>Start-trigger delay capabilities</td><td>Available data for post-trigger: 1 through 2^{30}</td></tr>
<tr><td>Stop-trigger delay capabilities</td><td>Available data for post-trigger: 1 through 2^{30}</td></tr>
<tr><td rowspan="4">Event notifications</td><td>The analog output is terminated.</td></tr>
<tr><td>The analog output is stopped.</td></tr>
<tr><td>The current-loop open failure is detected.</td></tr>
<tr><td>The reset input signal is asserted.</td></tr>
<tr><td rowspan="3">Data processings</td><td>Averaging (simple/shifted)</td></tr>
<tr><td>Interpolation</td></tr>
<tr><td>Data conversion: from binary to physical value and vice versa.</td></tr>
</table>

Notes:

- The maximum output update rate depends on the board specifications, operating environments, and other conditions.
- Each output update rate in the table is a single channel update rate. When two or more channels are output simultaneously, the rate may decrease depending on the number of channels.
- These values depend on the board specifications.

Chapter 3 Installation and Board Configuration

3.1 Installing the Linux Driver Software

1. Install the board into the open slot according to the manual came with the board.
2. Run Linux.
3. Install the Linux driver software according to the instructions of the installer.

```
#sh install
```

Please refer to the README.HTM for details of how to install the driver software.

3.2 Loading the Driver Modules

Load the GPH-3300 driver modules with `insmod`. The following shows an example for the kernel version 2.4.2.

```
#cd /lib/modules/2.4.2/misc
#insmod dpg0100.o
#insmod cp3300.o
```

Load the dpg0100.o, and then load the cp3300. You must follow this loading order.

3.3 Configuring the Device Numbers

1. Start the device number setting utility dpg0101.

```
#/usr/bin/dpg0101
```

2. When the device number setting utility starts, the following information and prompt will be displayed.

```
************************************************
 Setup Utility
------------------------------------------------
 Version: 1.01-02
------------------------------------------------
 Copyright 2002 Interface Corporation.
                  All rights reserved.
************************************************
Enter the model number of the product: GPG/GPH-
```

3. Enter 3300 and press the **Enter** key. The software searches every Interface analog output board installed on the system, then displays information about them.

```
=============================================
 Ref.ID | Type       | RSW1 | ADDA | Dev No.
 ---------------------------------------------
      1 | PCI/PAZ-3310  |    0 |  da   |     1
 ---------------------------------------------
      2 | PCI/PAZ-3325  |    1 |  da   |     2
 =============================================
```

Code	Description
Ref. ID	Reference ID of the board
Model	Module number of the board
RSW1	RSW1 setting value
ADDA	Available function (ad: analog input, da: analog output)
Device No.	Device number assigned to the board. This number is changeable.

4. Select the command.

```
************* Command ************
1. Change the Device Number.
2. Delete the Device Number.
3. Load new device setting file.
4. Run the initialization program.
99.Exit the program.
**********************************
Enter the command number:
```

No.	Command	Description
1	Change the device number	Changes the device number of the board.
2	Delete the device number	Deletes the device number. To delete it, enter the ID of the board.
3	Load new device setting file	Loads other device setting file.
4	Run the initialization program	The GPH-3300 doesn't support this command.
99	Exit the program	Exit the device number setting utility.

3.4 Programming

This section explains how to write the program to output one sample using the PCI/PAZ-3310 board whose RSW1 setting value is 0. When you use Kylix, refer to "5.4 Kylix."
After writing the program, save this file named as datest.c.

```
#include <stdio.h>
#include "fbida.h"

int main()
{
    int  nRet;
    DASMPLCHREQ  DaSmplChReq[2];
    unsigned short  Data[2];

    nRet = DaOpen(1);
    if(nRet != DA_ERROR_SUCCESS) printf("Failed to open the device.\n");
    else printf("The operation was successfully completed.\n");

    DaSmplChReq[0].ulChNo = 1;
    DaSmplChReq[0].ulRange = DA_5V;
    DaSmplChReq[1].ulChNo = 2;
    DaSmplChReq[1].ulRange = DA_5V;
    Data[0] = 0x8000;
    Data[1] = 0xC000;

    // One analog data output
    nRet = DaOutputDA(1, 2, &DaSmplChReq[0], &Data[0]);
    if(nRet != DA_ERROR_SUCCESS) printf("Failed to output data.\n");
    else printf("The operation was successfully completed.\n");

    nRet = DaClose(1);
    if(nRet != DA_ERROR_SUCCESS) printf("Failed to close the board.\n");
    else printf("The operation was successfully completed.\n");

    return 0;
}
```

3.5 Compiling the Program

Compile the program made in "3.4 Programming." Type the command as follows.

```
#gcc -o datest datest.c -lgph3300
```

3.6 Running the Program

Run the program as follows.

```
#./datest
```

Refer to "3.7 Data Acquisition Programming Technique," for more details of how to program.

3.7 Data Acquisition Programming Technique

3.7.1 Continuous Analog Output Update

First, specify a buffer size to store output data by using the DaSetBoardConfig function. Second, configure analog output update conditions of the board by using the DaSetSamplingConfig function. Finally, store the data into the output buffer of the board by using the DaSetSamplingData function. Then start continuous analog output update by using the DaStartSampling function. When you use the PCI/PAZ-3305 board, the DaSetMode function configures the board-specific functionality.

Example (C)

```
int DaOutput(int DeviceNo )
{
    int  nRet, i;
    DASMPLREQ  DaSmplConfig;
    unsigned short  SmplData[512][2];

    nRet = DaOpen(DeviceNo);

    // Specify a buffer size.
    nRet = DaSetBoardConfig(DeviceNo, 512, NULL, NULL, 0);

    DaSmplConfig.ulChCount = 2;
    DaSmplConfig.SmplChReq[0].ulChNo = 1;
    DaSmplConfig.SmplChReq[0].ulRange = DA_5V;
    DaSmplConfig.SmplChReq[1].ulChNo = 2;
    DaSmplConfig.SmplChReq[1].ulRange = DA_5V;
    DaSmplConfig.ulSamplingMode = DA_IO_SAMPLING;
    DaSmplConfig.fSmplFreq = 10000.0;
    DaSmplConfig.ulSmplRepeat = 1;
    DaSmplConfig.ulTrigMode = DA_FREE_RUN;
    DaSmplConfig.ulTrigPoint = DA_TRIG_START;
    DaSmplConfig.ulTrigDelay = 0;
    DaSmplConfig.ulEClkEdge = DA_LOW_EDGE;
    DaSmplConfig.ulTrigEdge = DA_LOW_EDGE;
    DaSmplConfig.ulTrigDI = 0;

    // Configure analog output update conditions.
    nRet = DaSetSamplingConfig(DeviceNo, &DaSmplConfig);

    // Prepare output data.
    // Store analog output data to SmplData[512][2].
    for(i = 0; i < 512 ; i++){
```

(Continued)

```
        SmplData[i][0] = i;
        SmplData[i][1] = 512 - i;
    }
    // Set the analog output data.
    nRet = DaSetSamplingData(DeviceNo, &SmplData[0][0], 512);

    // Start the analog output update.
    nRet = DaStartSampling(DeviceNo, FLAG_SYNC);

    nRet = DaClose(DeviceNo);

    return 0;
}
```

3.7.2 One-Shot Analog Output

The software-paced one analog output needs to use the DaOutputDA function.

Example (C)

```
int DaOutput(int DeviceNo)
{
    int  nRet;
    DASMPLCHREQ  DaSmplChReq[2];
    unsigned short  Data[2];

    nRet = DaOpen( DeviceNo );

    DaSmplChReq[0].ulChNo = 1;
    DaSmplChReq[0].ulRange = DA_5V;
    DaSmplChReq[1].ulChNo = 2;
    DaSmplChReq[1].ulRange = DA_5V;
    Data[0] = 0x800;
    Data[1] = 0xC00;

    // One analog output
    nRet = DaOutputDA( DeviceNo, 2, &DaSmplChReq[0], &Data[0] );

    nRet = DaClose(DeviceNo);

    return 0;
}
```

3.7.3 Parallel Analog Output Update

In the parallel analog output update configuration, a single master and one or more slave boards exist in the system. The following shows how to start the parallel analog output update.

1. Specify a buffer size to store output data for each board by using the DaSetBoardConfig function.
2. Configure analog output update conditions for each board by using the DaSetSamplingConfig function.
3. Store the data into the output buffer for each board by using the DaSetSamplingData function.
4. Call the DaSyncSampling function in the slave boards.
5. Call the DaSyncSampling function in the master board.

Then each board synchronously starts analog output update.

Example (C)

```
int DaOutput(int MasterNo, int Slave1No, int Slave2No)
{
    int  nRet, i;
    DASMPLREQ  DaConfigMaster;
    DASMPLREQ  DaConfigSlave1;
    DASMPLREQ  DaConfigSlave2;
    unsigned short  DataMaster[512][2];
    unsigned short  DataSlave1[512][2];
    unsigned short  DataSlave2[512][2];

    nRet = DaOpen(MasterNo);  // Master board
    nRet = DaOpen(Slave1No);  // Slave board 1
    nRet = DaOpen(Slave2No);  // Slave board 2

    // Specify a buffer size.
    DaSetBoardConfig( MasterNo, 512, NULL, NULL, 0 );
    DaSetBoardConfig( Slave1No, 512, NULL, NULL, 0 );
    DaSetBoardConfig( Slave2No, 512, NULL, NULL, 0 );

    DaConfigMaster.ulChCount = 2;
    DaConfigMaster.SmplChReq[0].ulChNo = 1;
    DaConfigMaster.SmplChReq[0].ulRange = DA_5V;
    DaConfigMaster.SmplChReq[1].ulChNo = 2;
    DaConfigMaster.SmplChReq[1].ulRange = DA_5V;
    DaConfigMaster.ulSamplingMode = DA_IO_SAMPLING;
    DaConfigMaster.fSmplFreq = 10000.0;
    DaConfigMaster.ulSmplRepeat = 1;
    DaConfigMaster.ulTrigMode = DA_FREE_RUN;
    DaConfigMaster.ulTrigPoint = DA_TRIG_START;
    DaConfigMaster.ulTrigDelay = 0;
    DaConfigMaster.ulEClkEdge = DA_LOW_EDGE;
```

(Continued)

```
DaConfigMaster.ulTrigEdge = DA_LOW_EDGE;
DaConfigMaster.ulTrigDI = 0;

// Configure the analog output update conditions (master board).
nRet = DaSetSamplingConfig( MasterNo, &DaConfigMaster );

DaConfigSlave1.ulChCount = 2;
DaConfigSlave1.SmplChReq[0].ulChNo = 1;
DaConfigSlave1.SmplChReq[0].ulRange = DA_5V;
DaConfigSlave1.SmplChReq[1].ulChNo = 2;
DaConfigSlave1.SmplChReq[1].ulRange = DA_5V;

// Configure the analog output update conditions (slave board 1).
nRet = DaSetSamplingConfig(Slave1No, &DaConfigSlave1);

DaConfigSlave2.ulChCount = 2;
DaConfigSlave2.SmplChReq[0].ulChNo = 1;
DaConfigSlave2.SmplChReq[0].ulRange = DA_5V;
DaConfigSlave2.SmplChReq[1].ulChNo = 2;
DaConfigSlave2.SmplChReq[1].ulRange = DA_5V;

// Configure the analog output update conditions (slave board 2).
nRet = DaSetSamplingConfig(Slave2No, &DaConfigSlave2);

// Prepare output data.
for(i = 0; i < 512 ; i++){
   DataMaster[i][0] = i;
   DataSlave1[i][0] = i;
   DataSlave2[i][0] = i;
   DataMaster[i][1] = 512 - i;
   DataSlave1[i][1] = 512 - i;
   DataSlave2[i][1] = 512 - i;
}

// Set the analog output data.
DaSetSamplingData(MasterNo, &DataMaster[0][0], 512);
DaSetSamplingData(Slave1No, &DataSlave1[0][0], 512);
DaSetSamplingData(Slave2No, &DataSlave2[0][0], 512);

// Parallel analog output update (slave boards)
nRet = DaSyncSampling(Slave2No, DA_SLAVE_MODE);
```

(Continued)

```
    nRet = DaSyncSampling(Slave3No, DA_SLAVE_MODE);

    // Parallel analog output update (master board)
    nRet = DaSyncSampling( MasterNo, DA_MASTER_MODE);

    return 0;
}
```

Note: You can use the same external signal to synchronize analog output update timing on each board. Refer to "3.7.4 External Trigger," and "3.7.5 External Clock," respectively.

3.7.4 External Trigger

The analog output update can start at the assertion of an external trigger. Configure the analog output update conditions by using the DaSetSamplingConfig function. The DaStartSampling function starts continuous analog output on the board.

Example (C)

```
int DaOutput(int DeviceNo)
{
    int       nRet, i;
    DASMPLREQ  DaSmplConfig;
    unsigned short  SmplData[512][2];

    nRet = DaOpen(DeviceNo);

    // Specify a buffer size.
    nRet = DaSetBoardConfig(DeviceNo, 512, NULL, NULL, 0);

    if(nRet != DA_ERROR_SUCCESS) return nRet;

    DaSmplConfig.ulChCount = 2;
    DaSmplConfig.SmplChReq[0].ulChNo = 1;
    DaSmplConfig.SmplChReq[0].ulRange = DA_5V;
    DaSmplConfig.SmplChReq[1].ulChNo = 2;
    DaSmplConfig.SmplChReq[1].ulRange = DA_5V;
    DaSmplConfig.ulSamplingMode = DA_IO_SAMPLING;
    DaSmplConfig.fSmplFreq = 10000.0;
    DaSmplConfig.ulSmplRepeat = 1;
    DaSmplConfig.ulTrigMode = DA_EXTTRG;  // External trigger
    DaSmplConfig.ulTrigPoint = DA_TRIG_START;
    // Start analog output update by the trigger.
    DaSmplConfig.ulTrigDelay = 0;
    DaSmplConfig.ulEClkEdge = DA_LOW_EDGE;
    DaSmplConfig.ulTrigEdge = DA_LOW_EDGE;
    DaSmplConfig.ulTrigDI = 0;

    // Configure the analog output update conditions.
    nRet = DaSetSamplingConfig(DeviceNo, &DaSmplConfig);

    // Prepare output data.
    // Store analog output data to SmplData [512][2].
    for(i = 0; i < 512 ; i++){
       SmplData[i][0] = i;
```

(Continued)

```
        SmplData[i][1] = 512 - i;
    }

    // Set the analog output data.
    nRet = DaSetSamplingData(DeviceNo, &SmplData[0][0], 512);

    // Wait for an assertion of the external trigger.
    // Start continuous analog output.
    nRet = DaStartSampling(DeviceNo, FLAG_SYNC);

    nRet = DaClose(DeviceNo);

    return 0;
}
```

3.7.5 External Clock

An external clock can be used as an analog output update pacer clock. To use the external clock, specify zero to analog output update rates. The DaStartSampling function starts continuous analog output update on the board.

Example (C)

```
int DaOutput(int DeviceNo)
{
    int  nRet, i;
    DASMPLREQ  DaSmplConfig;
    unsigned short  SmplData[512][2];

    nRet = DaOpen(DeviceNo);

    // Specify a buffer size.
    nRet = DaSetBoardConfig(DeviceNo, 512, NULL, NULL, 0);

    DaSmplConfig.ulChCount = 2;
    DaSmplConfig.SmplChReq[0].ulChNo = 1;
    DaSmplConfig.SmplChReq[0].ulRange = DA_5V;
    DaSmplConfig.SmplChReq[1].ulChNo = 2;
    DaSmplConfig.SmplChReq[1].ulRange = DA_5V;
    DaSmplConfig.ulSamplingMode = DA_IO_SAMPLING;
    DaSmplConfig.fSmplFreq = 0.0;      // Use the external clock.
    DaSmplConfig.ulSmplRepeat = 1;
    DaSmplConfig.ulTrigMode = DA_FREERUN;
    DaSmplConfig.ulTrigPoint = DA_TRIG_START;
    DaSmplConfig.ulTrigDelay = 0;
    DaSmplConfig.ulEClkEdge = DA_LOW_EDGE;
    DaSmplConfig.ulTrigEdge = DA_LOW_EDGE;
    DaSmplConfig.ulTrigDI = 0;

    // Configure the analog output update conditions.
    nRet = DaSetSamplingConfig( DeviceNo, &DaSmplConfig );

    // Store analog output data to SmplData [512][2].
    for(i = 0; i < 512 ; i++){
       SmplData[i][0] = i;
       SmplData[i][1] = 512 - i;
    }
    // Set the analog output data.
    nRet = DaSetSamplingData( DeviceNo, &SmplData[0][0], 512 );
```

(Continued)

```
    // Start the analog output update with the external clock pulses.
    nRet = DaStartSampling( DeviceNo, FLAG_SYNC );

    nRet = DaClose(DeviceNo);

    return 0;

}
```

Chapter 4 Functional Descriptions

4.1 Triggering

4.1.1 Trigger Overviews

Trigger signals decide when to start and/or stop the analog output. The tigger has the two following types:

- External trigger

- External trigger with mask using a general purpose digital input pin

Using the trigger delay capability, the timing when the analog output starts or stops changes depending on the number of the data output before or after the trigger.

4.1.2 External Trigger

The external trigger capability determines that the analog output starts or stops when an external signal is asserted.

- External Trigger Input Pin

It depends on the board specifications.

Data Transfer Mode	External Trigger Input Pin
Programmed I/O	EXINT IN
Memory	EXTRG IN

- External Trigger Input Configuration

Either rising or falling edge can be selected to assert a trigger. This capability depends on the board specifications, please refer to the user's manual of your board.

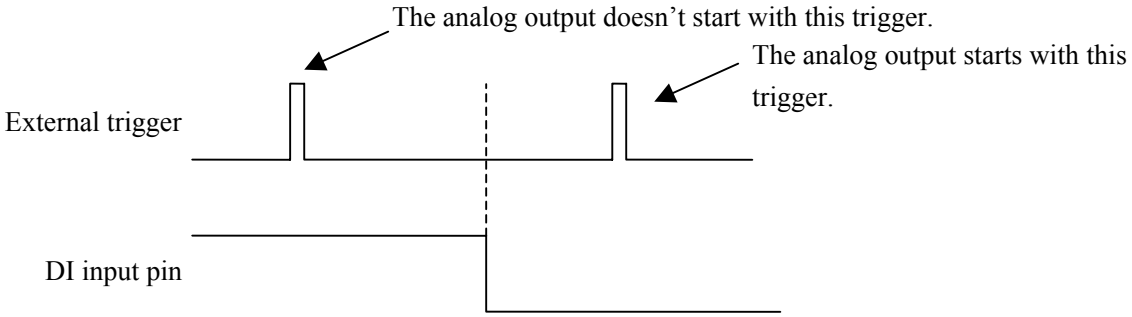
Using the trigger delay capability, the timing when the analog output starts or stops changes depending on the number of the data output before or after the trigger.

4.1.3 External Trigger with Mask Using a General Purpose Digital Input Pin

This capability adds the mask conditions to the analog trigger capability.
When the signal on the specified general purpose digital input pin (hereafter DI pin) is low level, the assertion of the external trigger signal is recognized as valid assertion.
In other words, if the signal on the DI pin is high level, the assertion of the external trigger signal is ignored, or masked.

The digital input pin depends on the board specifications.

Data Transfer Mode	Digital Input Pin
Programmed I/O	IN1 or IN2 selectable
Memory (PCI-3335 and PCI-3337)	IN1
Memory (PCI/PAZ-3305)	N/A



4.1.4 Trigger Delays

Using trigger delay capability, actual analog output starts or stops after a trigger is asserted. The GPH-3300 has the following trigger delay capability.

- Post-Trigger Delay

An actual start-point or stop-point is delayed by the time according to the number of post-trigger data.

4.2 Data Format

4.2.1 Data Type

The GPH-3300 handles the three following data types.

Data Type	Description
Analog output data	Data handled by the DaStartSampling and DaOutputDA functions.
Digital input data	Data handled by the DaInputDI function.
Digital output data	Data handled by the DaOutputDO function.

4.2.2 Analog Output Data

The following tables describe how the data are stored.

- Data storage format
- Data format in a frame
- Bit arrangement of data
- Data area size

- Data Storage Format

First analog output	Channel 1
	Channel 2
	:
	:
	Channel *n*
Second analog output	Channel 1
	Channel 2
	:
	:
	Channel *n*
:	:
:	:
M-th analog output	Channel 1
	Channel 2
	:
	:
	Channel *n*

- Data Format in a Frame

Channel 1
Channel 2
:
:
Channel *n*

Using the DaOutputDA function, only one frame of analog output data exists in the buffer.

- Bit Arrangement of Data

Resolution	Data
8 bits: one unsigned char (8 bit) used	bit7 . . . bit0
12 bits: one unsigned short (16 bit) used	bit15 . . . bit12 bit11 . . . bit0 0
16 bits: one unsigned short (16 bit) used	bit15 . . . bit0
24 bits: one unsigned long (32 bit) used	bit31 . . . bit24 bit23 . . . bit0 0

- Data Area Size

Required data area is given by the following equation.

Data area [byte] = (the number of channels) * (the number of data) * (data unit in bytes required for the resolution)

Where, data unit is as follows.

Resolution	Data Unit [byte]
8 bits	1
12 bits	2
16 bits	2
24 bits	4

Example)

For 4 channels, 100 data, and 16-bit resolution:

Required data area [byte] = 4 * 100 * 2 = 800

4.2.3 Digital Input Data

This data shows a status of general purpose digital input pins on the board. The polarity and the number of pins depend on the board specifications. Please refer to the user's manual.

- Bit Arrangement of Digital Input Data

	bit31 . . . bit16	bit15	. . .	bit0
Unsigned long (32 bits)	Not used	IN16	. . .	IN1

4.2.4 Digital Output Data

This data controls general purpose digital output pins on the board. The polarity and the number of pins depend on the board specifications. Please refer to the user's manual.

- Bit Arrangement of Digital Output Data

	bit31 . . . bit16	bit15	. . .	bit0
Unsigned long (32 bits)	Not used	OUT16	. . .	OUT1

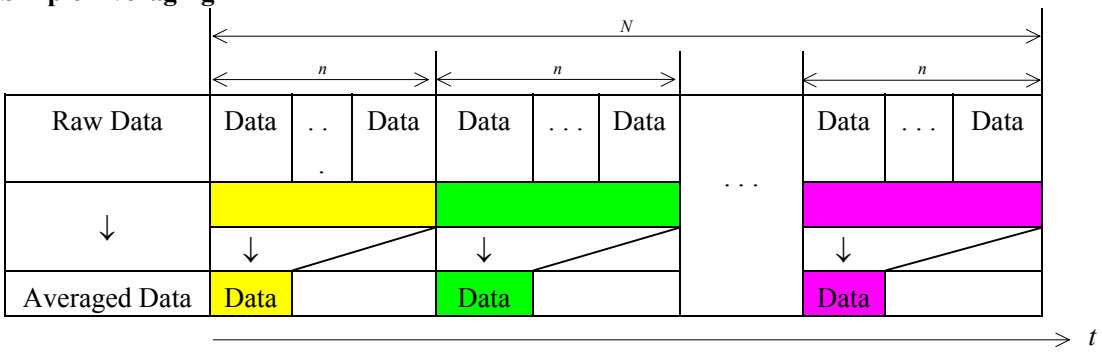
4.3 Averaging

Analog data can be averaged by using the DaDataConv function. The two following methods are supported.

- Simple averaging

- Shifted averaging

- Simple Averaging



After averaging *n* raw data by this method, the effective analog output rate and the number of effective data averaged are reduced to 1/*n* from their originals. Assume that the number of raw data is *N*, the number of effective data will be *N*/*n* after averaging. When *N* is not divisible by *n*, the remainder is not used for averaging. When you use the DaDataConv function, specify DA_CONV_AVERAGE1 and *n* for uEffect and uCount, respectively.

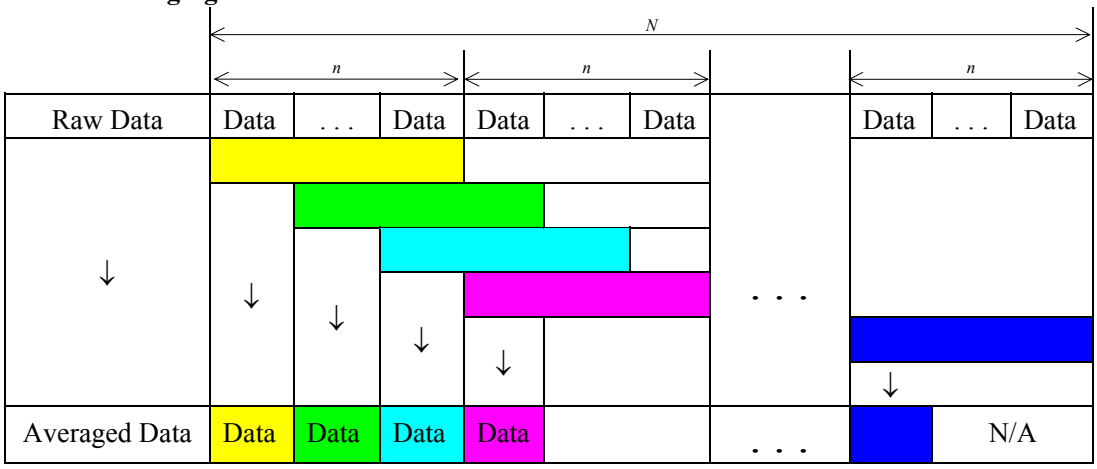
For the numerical formulas, please refer to the followings.

The data series before averaging: x_k , $k = 0, \ldots, (N\text{-}1)$

The data series after averaging: y_i , $i = 0, \ldots, \frac{N}{n}\text{-}1$

$y_i = \frac{1}{n} \sum_{k=ni}^{n(i+1)-1} x_k$ S: sigma

- Shifted Averaging



Assume that the number of raw data is *N*. After averaging *n* raw data, the number of effective data averaged is (*N* -*n* + 1). The analog output update rate is not changed. When you use the DaDataConv function, specify DA_CONV_AVERAGE2 and *n* for uEffect and uCount, respectively.

For the numerical formulas, please refer to the followings.

The data series before averaging: x_k, $k = 0, \ldots, (N\text{-}1)$

The data series after averaging: y_i, $i = 0, \ldots, (N\text{-}n)$

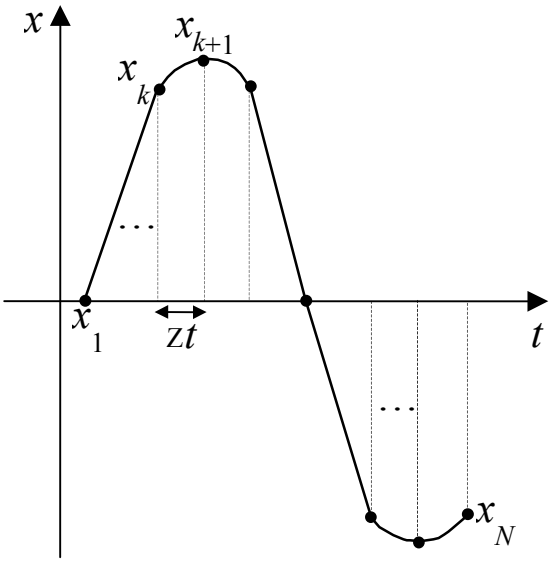
$y_i = \frac{1}{n} \sum_{k=i}^{i+n-1} x_k$ S: sigma

4.4 Interpolation

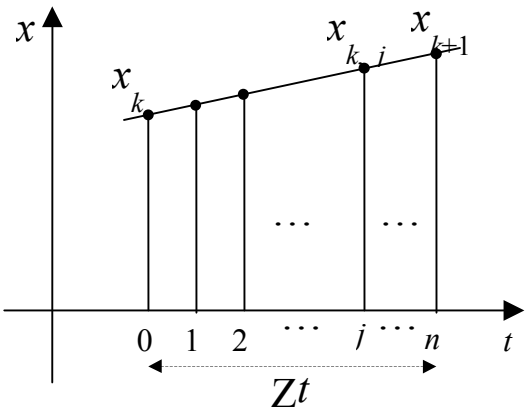
The GPH-3300 provides the interpolation capability to reduce quantization error by the digital-to-analog conversion. This product adopts the linear interpolation. When an interpolation parameter *n* is given, new data of (*n*-1) points are generated between two data sampled from a source waveform.
Where we assume that the sampled source waveform described by a series of data x_k (k = 1, 2, . . . , N), (N is the number of data that consist of the source waveform sampled), new points of data $x_{k,0}$ (= x_k), $x_{k,1}$ $x_{k,2}$, . . . , $x_{k,j}$, . . . , $x_{k,n}$ (= x_{k+1}) between x_k and x_{k+1} are calculated as follows:

$x_{k,j} = x_k + (x_{k+1} - x_k) * (j / n)$ where j = 0, 1, . . . , n.

As a result, the analog output rate of the new waveform results in *n*-times that of the source waveform. The new waveform consists of (Nn - n + 1) points of data. When you use the DaDataConv function, specify DA_CONV_SMOOTH and *n* for uEffect and uCount, respectively.



Zt: update interval



4.5 Waveform Generation Mode (applicable only to the PCI/PAZ-3305)

The PCI/PAZ-3305 supports the two following waveform generation modes.

- Time-based waveform generation: Waves are generated by using 1 microsecond of the base clock.
- Frequency-based waveform generation: Waves are generated by using 1 Hz of the base frequency.

Mode	Time-based waveform generation	Frequency-based waveform generation
Number of Data	1 through 524288	524288
Effective Number of Output Data	All	524288/(*n*-th power of two)
Update Rate	- 0 to 2.5 MHz - 5 MHz (fixed)	524288 Hz
Intervals between Waves	Supported	Not supported
Repetition	- 1 through 65536 - Infinite	Infinite

4.5.1 Time-Based Waveform Generation Mode

1. Generation Mode Selection

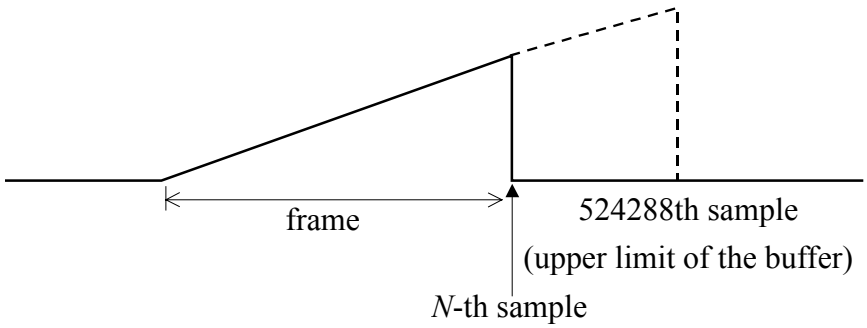
Two output modes are available in the time-based waveform generation mode.

- **Single output mode**
- **Repeat output mode**

Single Output Mode

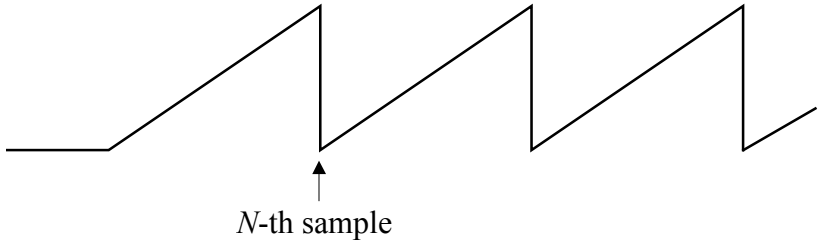
One cycle or one frame of waveform is output in this mode.

The following figure shows a waveform that consists of *N* samples.



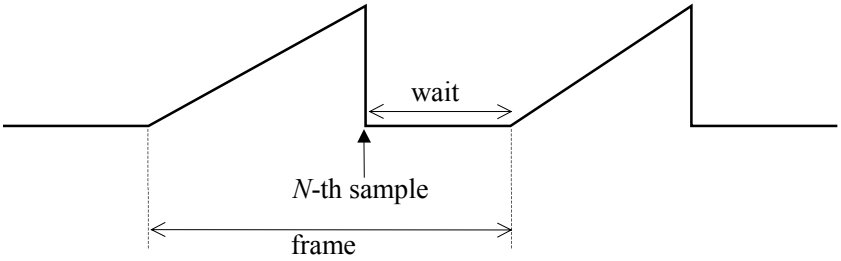
Repeat Output Mode

The waveform can be output repeatedly in this mode. The repetition is 1 through 65536 or infinite. If you specify infinite as the repetition, the output continuously performs until the DaStopSampling function is called.



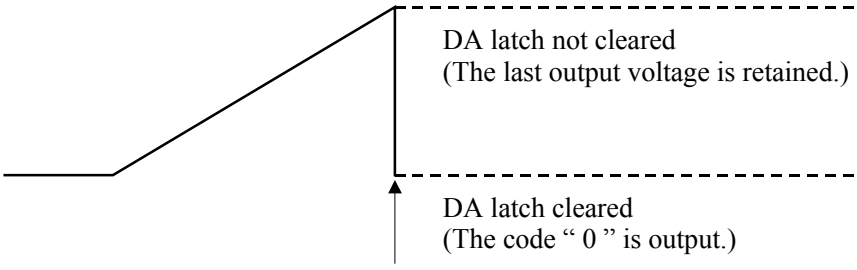
2. Variable Frame Cycle in the Repeat Output Mode

A wait state can be inserted between one waveform and the next waveform. A frame including the wait state is defined as follows. You can specify the frame frequency in the range of 0.01 Hz to 2.5 MHz.



3. DA Latching

When the waveform output is completed, the last output data is hold or set to the lowest value of the output range (negative full-scale: output code of 0000h) depending on the DA latch settings. The following figure shows the DA latching capability.



A waveform output is completed.

(Negative full scale value)

4. Update Pacer Clock Selection

You can select the update pacer clock in time-based waveform generation mode.

- Internal programmable timer: 0 MHz to 2.5 MHz (variable)
- Crystal: 5 MHz (fixed)

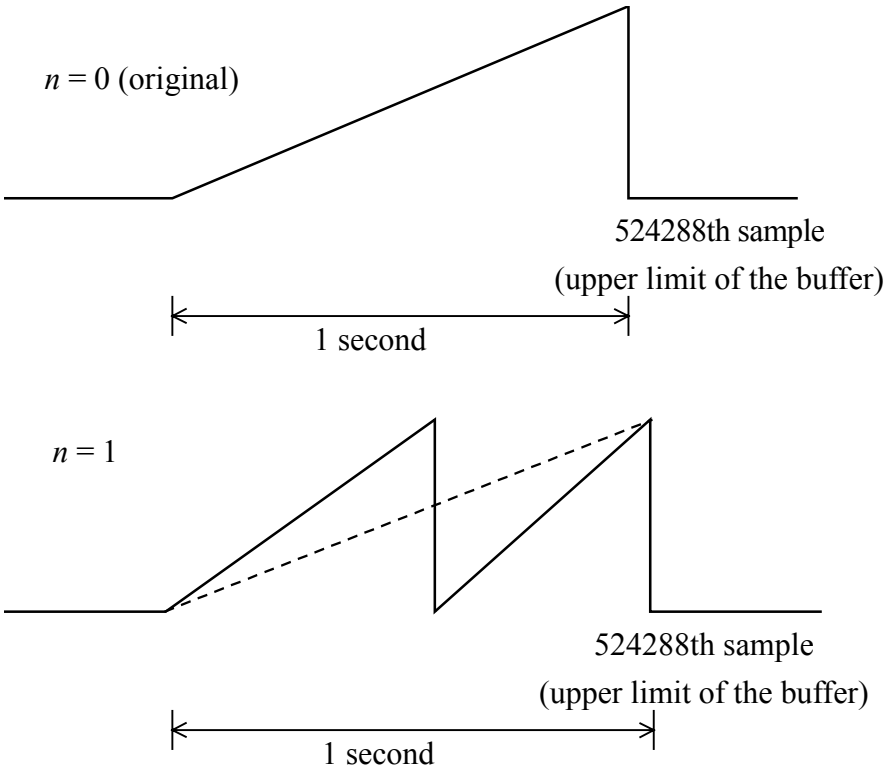
If 0 is specified as the update rate in the fSmplFreq member of the DASMPLREQ structure, an external update pacer clock is used.

4.5.2 Frequency-Based Waveform Generation Mode

In the frequency-Based waveform generation mode, the update pacer clock is fixed at 524288 Hz.

1. Frequency-Based Waveform Generation

A waveform that has a frequency given by the origianl frequency multiplied by the *n*-th power of two can be generated. The waveform data thinned out from source data are output. For example, one is specified as the power, the output data sequence is as follows: first data, third data, fifth data,. . .



2. Analog Output Counter

The frequency-based waveform generation mode allows to keep the counter or clear it when the output stops. When next output starts, the following output can start at previous kept counter value.

4.5.3 Independent Programmable Range Settings

The PCI/PAZ-3305 supports independent programmable voltage ranges. Each channel can be individually configured for output ranges. You can configure the ranges by using the DaSetMode function. The maximum voltage range is 10.0 V in software.

4.6 Attentions

4.6.1 Basic Flow of Programs

First, use the DaOpen function to initialize the board. After the initialization successfully completed, place your program codes as required. Before exiting your program, use the DaClose or DaCloseEx function to close the board and release resources properly.

4.6.2 External Clock, External Trigger, and Mask

Programmed I/O mode

Board	PCI		PAZ		CTP		
	3174 (DA)	3175 (DA)	3174 (DA)	3176 (DA)	3174 (DA)	3175 (DA)	3182
	3176 (DA)	3310	3310	3325	3325	3329	3338
	3325	3329	3329	3336	3340A	3340B	3340C
	3336	3338	3338	3340	3340D	3342	3343
	3340	3341A	3521(DA)		3346	3347	3348
	3342A	3343A			3349	3350	3351
	3345A	3346A			3521 (DA)	3522 (DA)	3523 (DA)
	3347	3521(DA)					
	3522A (DA)	3523A (DA)					

- A signal connected to the EXINT IN pin is handled as an external clock signal or an external trigger signal depending on the settings.
- If you use an external trigger with mask using a general purpose digital input pin, use the EXINT IN and IN1 pins or EXINT IN and IN2 pins for the mask condition.

FIFO mode

Board	PCI
	3525

- An external clock capability is not supported.
- A signal connected to the EXTRG IN pin is handled as an external trigger signal.
- A trigger delay capability is not supported.
- An external trigger with mask using a general purpose digital input pin capability is not supported.

Memory mode 1

Board	PCI	
	3335	3337

- A signal connected to the EXCLK IN pin is handled as an external clock signal.
- A signal connected to the EXTRG IN pin is handled as an external trigger signal.

Memory mode 2

Board	PCI	PAZ
	3305	3305

- A signal connected to the CN4 is handled as an external clock signal.
- A signal connected to the CN3 is handled as an external trigger signal.
- The direction, input or output, depends on the settings. Input and output cannot be specified at the same time.

When you use an external clock signal, set fSmplFreq to 0 by using the DaSetSamplingConfig function.

4.6.3 Attention to External Trigger

Board	PCI	
	3335	3337

If you specify the DA_EXTTRG parameter with the DA_TRIG_STOP or DA_TRIG_START_STOP parameters in the DASMPLREQ structure, the number of analog output data should be 524288.

4.6.4 Reset in Capability

Board	PCI				PAZ	
	3310	3335	3336	3337	3310	3336

When a signal is asserted on the RESET IN pin, the analog output ranges of all channels are reinitialized to unipolar 0 V to 5 V. The output voltage of each channel is set to 0 V.

Board	PCI	PAZ
	3340	3340

When a signal is asserted on the RESET IN pin, the output voltage of each channel is set to 0 V.

Board	CTP			
	3340A	3340B	3340C	3340D

When a signal is asserted on the RESET IN pin, the output voltage of each channel is set to 0 V.

Board	PCI	CTP
	3347	3347

When a signal is asserted on the RESET IN pin, the output voltage of each channel is set to 0 V. Software can generate an event during analog output.

4.6.5 Handling Data Greater than Buffer Memory Size

Board	PCI	
	3335	3337

The driver software can handle analog output data greater than on-board buffer memory size at a time. But, if the number of analog output data is greater than on-board buffer memory size, the output cannot repeat.

4.6.6 Current-Loop Open Failure Event

Board	PCI	PAZ	CTP
	3325	3325	3325

When a current-loop open failure is detected on any output channels, an interrupt occurs. Then a current-loop open failure event is signaled. At this moment, DA_EVENT_CURRENT_OFF of event source is set in the ulSmplEventFactor argument of the DaGetBoardConfig function. The output channel on which the current-loop open failure was detected cannot be determined by software.

4.6.7 Range Configuration

Board	PCI	PAZ
	3305	3305

Before one analog data is output by the DaOutputDA function, you need to configure the range by using the DaSetSamplingConfig or DaSetMode functions.

4.6.8 Number of Analog Output Data

Board	PCI	PAZ
	3305	3305

The number of analog output data you can specify is 1 through 524288.

4.6.9 External Clock Output

Board	PCI			PAZ		CTP	
	3310	3335	3336	3310	3336	3340A	3340B
	3337	3340		3340		3340C	3340D

The EXCLK OUT pin outputs an internal analog update pacer clock depending on the settings. The DaSetMode function configures this feature. To output an internal update pacer clock signal through the EXCLK OUT pin, specify DA_EXCLK_OUT to the ulExClock member of the DAMODEREQ structure. If you specify DA_EXCLK_IN, the EXCLK OUT pin disconnects the clock signal and output high level signal.

4.6.10 Genelar Purpose Digital Input/Output

Board	PCI		PAZ	CTP	
	3174 (DA)	3175 (DA)	3174 (DA)	3174 (DA)	3175 (DA)
	3176 (DA)		3176 (DA)	3182 (DA)	

The general purpose digital input/output pins cannot be controled by using the DaInputDI and DaOutputDo functions.

4.6.11 Analog Output Update Rate Limitation

Board	PCI		PAZ	CTP		
	3305	3310	3305	3325	3329	3338
	3325	3329	3310	3340A	3340B	3340C
	3335	3336	3325	3340D	3342	3343
	3337	3338	3329	3346	3347	3348
	3340	3341A	3336	3349	3350	3351
	3342A	3343A	3338	3521 (DA)	3522 (DA)	3523 (DA)
	3345A	3346A	3340			
	3521 (DA)	3522A (DA)	3521 (DA)			
	3523A (DA)					

The analog output update rate is configurable by the DaSetSamplingConfig function, and the rate is obtained by the following equation:

$$f = \frac{f0}{N0 * N1}.$$

Where,

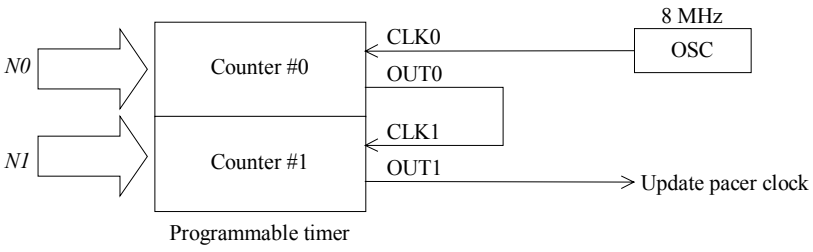
f: Update rate (Hz)

f0: Base clock frequency (Hz), 8 MHz

N0: Integer divisor of the counter, 2 through 65535

N1: Integer divisor of the counter, 2 through 65535

The actual analog output update rate has some errors against the specified rate to the DaSetSamplingConfig function because the ideal combination of integer divisors *N0* and *N1* could not be obtained for the rate.



Example)
When an analog output update rate of 300 Hz is specified to the DaSetSamplingConfig function, appropriate integer divisors of *N0* and *N1* are 3 and 8889, respectively. And actual update rate becomes 299.99625 Hz.

$$\frac{8000000}{2 * 13334} = \frac{8000000}{26668} = 299.9850008 .$$

$$\frac{8000000}{3 * 8889} = \frac{8000000}{26667} = 299.9962500 .$$

$$\frac{8000000}{2 * 13333} = \frac{8000000}{26666} = 300.007502 .$$

4.6.12 Analog Output Update Condition

Board	PCI
	3525 (DA)

You can configure the analog output update conditions of the board by using the DaSetSamplingConfig or DaSetFifoConfig function. In each case of using these functions, the last configuration condition is valid.

The following table shows the correspondence of the members of the DASMPLREQ structure to the DAFIFOREQ structure.

DASMPLREQ Structure	DAFIFOREQ Structure
ulChCount	ulChCount
SmplChReq	SmplChReq
ulSamplingMode	-
fSmplFreq	fSmplFreq
ulSmplRepeat	ulSmplRepeat
ulTrigMode	ulStartTrigCondition, ulStopTrigCondition
ulTrigPoint	ulStartTrigCondition, ulStopTrigCondition
ulTrigDelay	0
ulEClkEdge	ulEClkEdge
ulTrigEdge	ulTrigEdge
ulTrigDI	0
0	ulSmplNum

Chapter 5 Reference

5.1 List of Functions

No.	Function	Description	Applicable Note
1	DaOpen	Opens a board and enables to access to the board.	-
2	DaClose	Closes a board and releases the resources. Any subsequent accesses to the board are forbidden.	-
3	DaCloseEx	Closes a board and releases the resources. Any subsequent accesses to the board are forbidden. In addition, the analog output status after closing the board is selectable by the parameter.	-
4	DaGetDeviceInfo	Retrieves specifications of the board.	-
5	DaSetBoardConfig	Configures event handling of the board	-
6	DaGetBoardConfig	Retrieves an event source on the board.	-
7	DaSetSamplingConfig	Configures analog output update conditions of the board.	-
8	DaGetSamplingConfig	Retrieves analog output update conditions of the board.	-
9	DaSetMode	Configures the board-specific functionality.	1
10	DaGetMode	Retrieves configuration information of the board-specific functionality.	1
11	DaSetSamplingData	Stores data into the output buffer of the board.	-
12	DaClearSamplingData	Clears the data in the output buffer.	-
13	DaStartSampling	Starts the analog output update on the board.	-
14	DaStartFileSampling	Reads data from a data file and outputs them to the board.	-
15	DaSyncSampling	Enables you to achieve a synchronous analog output update on boards connected in parallel.	2
16	DaStopSampling	Stops the analog output update performed as the overlapped operation.	-
17	DaGetStatus	Retrieves the analog output update status of the board.	-
18	DaSetOutputMode	Enables or disables the simultaneous analog output.	3
19	DaGetOutputMode	Retrieves the configuration of the simultaneous analog output.	3
20	DaOutputDA	Outputs one analog data on the board.	-
21	DaInputDI	Reads general purpose digital input pins on the board.	-

(Continued)

No.	Function	Description	Applicable Note
22	DaOutputDO	Writes data to general purpose digital output pins on the board.	-
23	DaSetFifoConfig	Configures analog output update conditions of the board at FIFO data transfer mode.	4
24	DaGetFifoConfig	Retrieves analog output update conditions of the board at FIFO data transfer mode.	4
25	DaSetInterval	Configures the inteval timer.	4
26	DaGetInterval	Retrieves the interval timer cycle.	4
27	DaSetFunction	Configures the function of the CN3 connector.	4
28	DaGetFunction	Retrieves the functional configuration of the CN3 connector.	4
29	DaDataConv	Converts forms of the analog data. Averaging and interpolation can be done with the conversion.	-
30	DaWriteFile	Writes data to the file from the buffer. Binary and CSV formats are supported.	-
31	fnConv	Is a placeholder for a callback routine used in the DaDataConv function. This function is called when each data is converted.	-
32	CallBackProc	Is a placeholder for a callback routine. This function is called when the analog output is completed.	-

Notes:

1: These functions are applicable to the PCI/PAZ-3305, PCI/PAZ-3310, PCI-3335, PCI/PAZ-3336, PCI-3337, and PCI/PAZ-3340.

2: This function is applicable to the PCI/PAZ-3310, PCI/PAZ-3329, PCI/PAZ-3336, PCI/PAZ-3340, PCI-3341A, PCI-3342A, PCI-3343A, PCI-3345A, PCI-3346A, PCI-3347, PCI/PAZ-3521, PCI-3522A, and PCI-3523A.

3: These functions are applicable to the PCI/PAZ-3329, PCI/PAZ-3338, CTP-3329, and CTP-3338.

4: These functions are applicable only to the PCI-3525.

5.1.1 DaOpen

The DaOpen function opens a board and enables to access to the board.

```
int DaOpen(
  int       nDevice
);
```

Parameter

nDevice Specifies the device number to open.

Return Value

The DaOpen function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_ALREADY_OPEN
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_OPEN
- DA_ERROR_USED_AD

Example

```
int nRet;

nRet = DaOpen(1);
```

Open the board whose device number is 1.

5.1.2 DaClose

The DaClose function closes a analog output board and releases the resources. Any subsequent accesses to the board are forbidden.

```
int DaClose(
  int       nDevice
);
```

Parameter

nDevice Specifies the device number opened by the DaOpen function.

Return Value

The DaClose function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following code. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER

Comments

- If you access to the board again, reopen it to call the DaOpen function.
- If this function is called while an output is running, the output is terminated.
- After closing the board, output voltage on every analog output channel is set to 0 V.

Example

```
int nRet;

nRet = DaOpen(1);
if(!nRet){
    :
    :
    nRet = DaClose(1);
}
```

Close the board whose device number is 1.

5.1.3 DaCloseEx

The DaCloseEx function closes a board and releases the resources. Any subsequent accesses to the board are forbidden. In addition, the analog output status after closing the board is selectable by the parameter.

```
int DaCloseEx(
  int       nDevice,
  int       nFinalState
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

nFinalState Specifies the output status after closing the board.

Code	Description
DA_OUTPUT_RESET	Resets the analog output status to the default settings.
DA_OUTPUT_MAINTAIN	Maintains the analog output status including the output code and the output range when the board is closed.

Return Value

The DaCloseEx function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following code. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_INVALID_PARAMETER

Comments

- If you access to the board again, reopen it to call the DaOpen function.
- If this function is called while an output is running, the output is terminated.
- To close the board, use either the DaClose function or DaCloseEx function depending on the purpose.
- If the board is closed with specifying DA_OUTPUT_MAINTAIN, the board is set to the last state when the DaOpen function is called.
- When the PCI-3525 is closed by the DaCloseEx function supplied with DA_OUTPUT_MAINTAIN, the CN4 connector on the board is set to the analog output mode (AOUT). In this case, any functions provided by GPH-3100 cannot control CN4. To resolve this situation, reopen PCI-3525 by the DaOpen function, and then close the board by the DaClose function.

Example

```
int nRet;

nRet = DaOpen(1);
if(!nRet){
    :
    :
    nRet = DaCloseEx(1, DA_OUTPUT_MAINTAIN);
}
```

Close the board whose device number is 1, and the output status is maintained.

5.1.4 DaGetDeviceInfo

The DaGetDeviceInfo function retrieves specifications of the board.

```
int DaGetDeviceInfo(
  int            nDevice,
  PDABOARDSPEC   pBoardSpec
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

pBoardSpec Points to the DABOARDSPEC structure to receive the specifications of the board.

Return Value

The DaGetDeviceInfo function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NULL_POINTER

Example

```
int nRet;
DABOARDSPEC BoardSpec;

nRet = DaOpen(1);
if(!nRet){
    nRet = DaGetDeviceInfo(1, &BoardSpec);
    if(!nRet)printf("Board model: %d\n", BoardSpec.ulBoardType);
}
```

Retrieve the specifications of the board whose device number is 1.

5.1.5 DaSetBoardConfig

The DaSetBoardConfig function configures event handling of the board.

```
int DaSetBoardConfig(
  int             nDevice,
  unsigned long*  ulSmplBufferSize,
  void*           pReserved,
  PLPDACALLBACK   pCallBackProc,
  int             nReserved
);
```

Parameters

nDevice	Specifies the device number opened by the DaOpen function.
ulSmplBufferSize	Specifies a size of the buffer to store output data. The default value is 1024.
pReserved	Reserved. Specify NULL.
pCallBackProc	Specifies an address of user callback routine to be called when the analog output stops. If you don't use a callback routine, specify NULL in C. The default setting is NULL.
nReserved	Reserved. Specify 0.

Return Value

The DaSetBoardConfig function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_ILLEGAL_PARAMETER

Comments

- The data stored in the buffer are cleared if the output buffer size is changed.
- The buffer size cannot be changed while analog output is running.
- The syntax of the callback routine is as follows in C. Please refer to the CallBackProc function.

```
void CALLBACK CallBackProc(int nReserved);
```

Example

```
int nRet;
unsigned long ulSmplBufferSize;

void CALLBACK CallBackProc(int dummy){
    :
}

ulSmplBufferSize = 2048;
nRet = DaSetBoardConfig(1, ulSmplBufferSize, NULL, CallBackProc, 0);
```

Set event handling on the board whose device number is 1.

5.1.6 DaGetBoardConfig

The DaGetBoardConfig function retrieves an event source on the board.

```
int DaGetBoardConfig(
  int           nDevice,
  unsigned long *ulSmplBufferSize,
  unsigned long *ulSmplEventFactor
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulSmplBufferSize Points to a variable to receive the output buffer size.

ulSmplEventFactor Points to a variable to receive an event source of the analog output.

Code	Description
DA_EVENT_STOP_TRIGGER	The analog output has been stopped because a trigger is asserted.
DA_EVENT_STOP_FUNCTION	The analog output has been stopped by software.
DA_EVENT_STOP_SAMPLING	The analog output terminated.
DA_EVENT_RESET_IN	The reset input signal is asserted.
DA_EVENT_CURRENT_OFF	The power failure has been detected.

Return Value

The DaGetBoardConfig function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following code. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NULL_POINTER

Example

```
int nRet;
unsigned long ulBufferSize, ulEventFactor;

void event_proc(int dummy)
{
    printf("Analog outputs completed.\n");
    nRet = DaGetBoardConfig(1, &ulBufferSize, &ulEventFactor);
    if(!nRet){
        printf("Buffer Size: %lXn", ulBufferSize);
        printf("Source: %lX\n", ulEventFactor);
    }
}

nRet = DaSetBoardConfig( 1, 100, NULL, event_proc, 0 );
if(!nRet){
       nRet = DaStartSampling(1, FLAG_ASYNC);
}
```

Retrieve an event source on the board whose device number is 1.

5.1.7 DaSetSamplingConfig

The DaSetSamplingConfig function configures analog output update conditions of the board.

```
int DaSetSamplingConfig(
  int           nDevice,
  PDASMPLREQ    pDaSmplConfig
);
```

Parameters

nDevice	Specifies the device number opened by the DaOpen function.
pDaSmplConfig	Points to the DASMPLREQ structure.

Return Value

The DaSetSamplingConfig function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_ILLEGAL_PARAMETER
- DA_ERROR_NULL_POINTER
- DA_ERROR_NOT_ALLOCATE_MEMORY

Comment

The number of analog output channels and repetitions cannot be changed while the analog output is running.

Example

```
int nRet;
DASMPLREQ DaSmplConfig;

DaSmplConfig.ulChCount = 2;
DaSmplConfig.SmplChReq[0].ulChNo = 1;
DaSmplConfig.SmplChReq[1].ulChNo = 2;

nRet = DaSetSamplingConfig(1, &DaSmplConfig);
```

Configure the analog output update conditions of the board whose device number is 1.

5.1.8 DaGetSamplingConfig

The DaGetSamplingConfig function retrieves analog output update conditions of the board.

```
int DaGetSamplingConfig(
  int           nDevice,
  PDASMPLREQ    pDaSmplConfig
);
```

Parameters

nDevice	Specifies the device number opened by the DaOpen function.
pDaSmplConfig	Points to the DASMPLREQ structure to receive analog output update conditions.

Return Value

The DaGetSamplingConfig function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NULL_POINTER

Comment

The default settings of the DASMPLREQ structure can be retrieved by calling this function immediately after opening the board.

Example

```
int nRet;
unsigned long i;
DASMPLREQ DaSmplConfig;

nRet = DaGetSamplingConfig(1, &DaSmplConfig);
if(!nRet){
    if(i=0; i<DaSmplConfig.ulChCount; i++{)
        printf("Output channel: %d\n",DaSmplConfig.SmplChReq[i].ulChNo);
    }
}
```

Retrieve the analog output update conditions of the board whose device number is 1.

5.1.9 DaSetMode

The DaSetMode function configures the board-specific functionality.

Model	Description
PCI/PAZ-3305	Configures the waveform generation mode.
PCI-3335, PCI-3337	Enables or disables the external trigger output through the EXTRG OUT pin.
PCI/PAZ-3310, PCI-3335, PCI/PAZ-3336, PCI-3337, PCI/PAZ-3340	Enables or disables the external clock output through the EXCLK OUT pin.

```
int DaSetMode(
  int            nDevice,
  PDASMPLREQ     pDaMode
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

pDaMode Points to the DAMODEREQ structure.

Return Value

The DaSetMode function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_ILLEGAL_PARAMETER
- DA_ERROR_NULL_POINTER

Comments

- The DaSetMode function isn't available while analog output is running.
- Please refer to the user's manual of the board and Waveform Generation Mode for details.
- For the PCI/PAZ-3310, PCI-3335, PCI/PAZ-3336, PCI-3337, and PCI/PAZ-3340, specify DA_EXCLK_IN to the ulExClock member of the DAMODEREQ structure to disable the external clock output through the EXCLK OUT pin.
- For the PCI-3335 and PCI-3337, specify DA_EXTRG_IN to the ulExControl member of the DAMODEREQ structure to disable the external trigger output through the EXTRG OUT pin.

Example

```
int nRet;
DAMODEREQ DaMode;

nRet = DaGetMode(1, &DaMode);
if(!nRet){
    DaMode.ulPulseMode = DA_MODE_SYNTHE;
    nRet = DaSetMode(1, &DaMode);
}
```

Configure the board-specific parameters of the board whose device number is 1.

5.1.10 DaGetMode

The DaGetMode function retrieves configuration information of the board-specific functionality.

Model	Description
PCI/PAZ-3305	Retrieves the waveform generation mode.
PCI-3335, PCI-3337	Retrieves the operation mode of the EXTRG OUT pin.
PCI/PAZ-3310, PCI-3335, PCI/PAZ-3336, PCI-3337, PCI/PAZ-3340	Retrieves the operation mode of the EXCLK OUT pin.

```
int DaGetMode(
  int            nDevice,
  PDASMPLREQ     pDaMode
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

pDaMode Points to the DAMODEREQ structure to receive board-specific configurations.

Return Value

The DaGetMode function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_ILLEGAL_PARAMETER
- DA_ERROR_NULL_POINTER

Comments

- For the PCI/PAZ-3305, the DAMODEREQ structure contains the waveform generation parameters.
- For the PCI-3335 and PCI-3337, the ulExControl member of the DAMODEREQ structure contains the mode of the EXTRG OUT pin: DA_EXTRG_IN (trigger output disabled) or DA_EXTRG_OUT (trigger output enabled).
- For the PCI/PAZ-3310, PCI-3335, PCI/PAZ-3336, PCI-3337, and PCI/PAZ-3340, the ulExClock member of the DAMODEREQ structure contains the mode of the EXCLK OUT pin: DA_EXCLK_IN or DA_EXCLK_OUT.

Example

```
int nRet;
DAMODEREQ DaMode;

nRet = DaGetMode(1, &DaMode);
if(!nRet){
    DaMode.ulPulseMode = DA_MODE_SYNTHE;
    nRet = DaSetMode(1, &DaMode);
}
```

Retrieve the board-specific parameters of the board whose device number is 1.

5.1.11 DaSetSamplingData

The DaSetSamplingData function stores data into the output buffer of the board.

```
int DaSetSamplingConfig(
  int           nDevice,
  void*         pSmplData,
  unsigned long ulSmplDataNum
);
```

Parameters

nDevice	Specifies the device number opened by the DaOpen function.
pSmplData	Points to the application buffer containing data to be transferred into the output buffer.
ulSmplDataNum	Specifies the number of data.

Return Value

The DaSetSamplingData function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_INVALID_PARAMETER
- DA_ERROR_NULL_POINTER
- DA_ERROR_SET_DATA

Comments

- This function only stores the data into the output buffer.
- To start analog output, please call the DaStartSampling function.
- If you call this function while the analog output is running, the data are set to the output buffer as the next output data. The new data will be output after the output of the previously set data is completed. In this case, once output of new data starts, old data no longer are needed.
- This function appends new data at the tail of the existing data in the output buffer except the PCI-3335 and PCI-3337. For the PCI-3335 and PCI-3337, all data stored in the output buffer are discarded and the new data are stored from the top of the output buffer.
- For the PCI/PAZ-3305, this function isn't available while analog output is running.

Example

```
int i, nRet;
unsinged shot SmplData[4096][2];

// Prepare output data.
for(i = 0; i < 4096; i++){
    SmplData[i][0] = i;
    SmplData[i][1] = 4095 - i;
}

// Set the analog output data into the output buffer.
nRet = DaSetSamplingData(1, &wSmplData[0][0], 4096);
```

Store 4096 data for each of 2 channels into the buffer of the board whose device number is 1.

5.1.12 DaClearSamplingData

The DaClearSamplingData function clears the data in an output buffer.

```
int DaClearSamplingData(
  int            nDevice
);
```

Parameter

nDevice Specifies the device number opened by the DaOpen function.

Return Value

The DaClearSamplingData function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following code. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER

Example

```
int nRet;

nRet = DaClearSamplingData(1);
```

Clear the data in the output buffer on the board whose device number is 1.

5.1.13 DaStartSampling

The DaStartSampling function starts an analog output update on the board.

```
int DaStartSampling(
  int           nDevice,
  unsigned long ulSyncFlag
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulSyncFlag Specifies whether the analog output update process is performed as an overlapped operation or not.

Flag	Description
FLAG_SYNC	Specifies that the analog output update is performed as a non-overlapped operation.
FLAG_ASYNC	Specifies that the analog output update is performed as an overlapped operation.

Return Value

The DaStartSampling function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_START_SAMPLING
- DA_ERROR_INVALID_PARAMETER

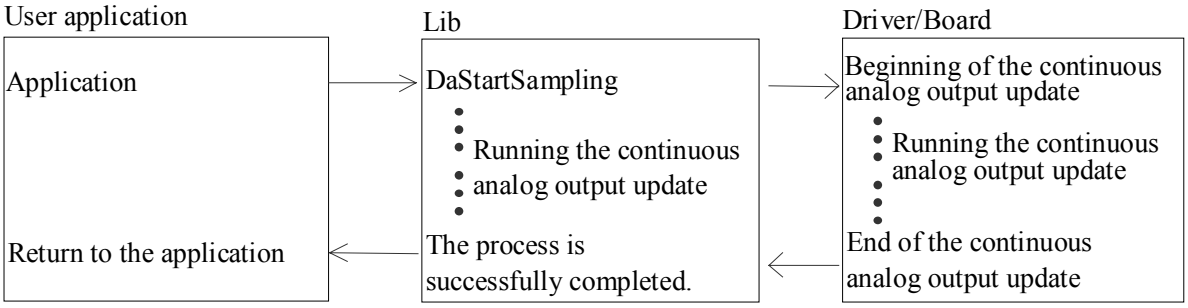
Comments

The analog output update stops when updating all analog output data specified by the DaSetSamplingData function are completed. If you choose the repetition, the analog output update stops when the specified repetitions are completed.

1. Overlapped/non-overlapped operations

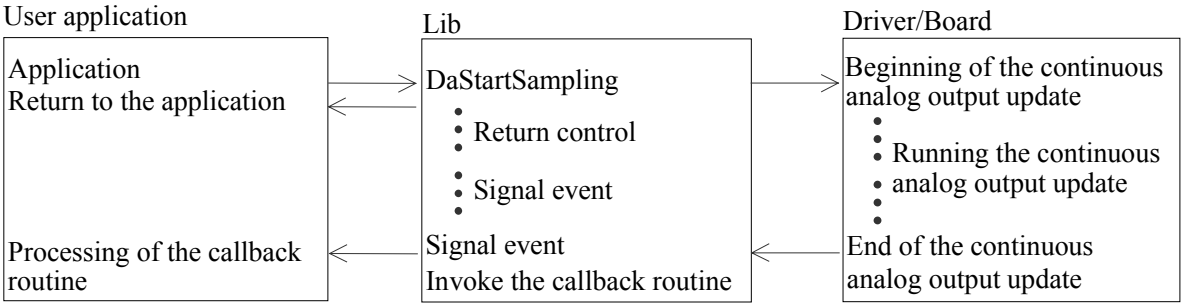
- Non-overlapped operation (FLAG_SYNC)

An applications wait until the continuous analog output update is completed.



- Overlapped Operation (FLAG_ASYNC)

Control returns immediately without waiting for the completion of the continuous analog output update.



Completion of the continuous analog output update is notified by the event signaling.

2. The overlapped continuous analog output update can be aborted by the DaStopSampling function.

3. Zero cannot be specified to the repetition count of the ulSmplRepeat member in the DASMPLREQ structure for non-overlapped analog output update operation.

Example

```
int i, nRet;
unsinged short SmplData[4096][2];
unsinged long Status, Count, AvailCount, AvailRepeat;

nRet = DaClearSamplingData(1);

// Prepare output data.

for (i = 0; i < 4096; i++) {
     SmplData[i][0] = i;
     SmplData[i][1] = 4095 - i;
}

// Set the analog output data into the output buffer.
nRet = DaSetSamplingData(1, &SmplData[0][0], 4096);
if(!nRet){
     nRet = DaStartSampling(1, FLAG_SYNC);
}

// Start the analog output as a non-overlapped operation on the
// board whose device number is 1.

nRet = DaStartSampling(1, FLAG_ASYNC);
if(!nRet){
do{
   nRet = DaGetStatus(1, &Status, &Count, &AvailCount, &AvailRepeat);
   if(nRet){
      printf("Status Error\n");
      DaClose(1);
      exit(1);
   }
 }while(Status != DA_STATUS_STOP_SAMPLING);
}
```

Start the analog output update as the overlapped operation on the board whose device number is 1.

5.1.14 DaStartFileSampling

The DaStartFileSampling function reads data from a data file and outputs them to the board.

```
int DaStartFileSampling(
  int           nDevice,
  char*         szPathName,
  unsigned long ulFileFlag,
  unsigned long ulSmplNum
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

szPathName Specifies the data file containing the output data.

ulFileFlag Specifies the format of the data file.

Flag	Description
FLAG_BIN	Binary format
FLAG_CSV	CSV format (physical value)

ulSmplNum Specifies the number of data.

Return Value

The DaStartFileSampling function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_START_SAMPLING
- DA_ERROR_INVALID_PARAMETER
- DA_ERROR_FILE_OPEN
- DA_ERROR_FILE_CLOSE
- DA_ERROR_FILE_READ

Comments

- If you use a CSV format file, the output update may not operate at the specified rate, because the data conversions from physical values to binary values need more time or overhead.
- Repetition isn't available.
- All of data previously existed in the output buffer are deleted when the analog output is started by the DaStartFileSampling function.

Example

```
int nRet;

nRet = DaStartFileSampling( 1, "test.dat", FLAG_CSV, 1024 );
```

Start the analog output on the board whose device number is 1 with reading the data from "test.dat".

5.1.15 DaSyncSampling

The DaSyncSampling function enables you to achieve a synchronous analog output update on boards connected in parallel.

A single master board distributes its internal analog output update pacer clock signal to other slave boards for the concurrent update without practical phase delay.

Calling the DaSyncSampling function on each slave board place the slave board into standby state for parallel update. Calling the DaSyncSampling function on a master board starts the simultaneous update on it with the waiting slave boards.

The DaSyncSampling function is applicable to the boards that have parallel simultaneous analog output update capability and they are listed below. The parallel update is not supported between different type of boards. You should connect the same type boards by the synch-cables.

Model		
PCI/PAZ-3310	PCI/PAZ-3329	PCI/PAZ-3336
PCI/PAZ-3340	PCI-3341A	PCI-3342A
PCI-3343A	PCI-3345A	PCI-3346A
PCI-3347	PCI/PAZ-3521	PCI-3522A
PCI-3523A		

```
int DaSyncSampling(
  int           nDevice,
  unsigned long ulMode
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulMode Specifies a role of the board, master, or slave by using the following codes exclusively.

Code	Description
DA_MASTER_MODE	Master
DA_SLAVE_MODE	Slave

Return Value

The DaSyncSampling function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_START_SAMPLING
- DA_ERROR_INVALID_PARAMETER

Comments

- The DaSyncSampling function always performs as an overlapped operation.
- The driver software is capable of event signaling and callback of your procedure at completion of the parallel analog output update. You should configure event settings and register your callback routine to the master board, not to slave boards.
- Use the DaStopSampling function to terminate the analog output update in progress.
- Use the DaSetSamplingConfig and DaSetMode functions to setup analog output update conditions.
- The driver software uses an output update rate of the master board in the parallel analog output update.
- Configure output ranges and output configurations for each channel on each board.
- When you select the programmed I/O mode as a data transfer mode on each board connected, you should configure each board to be the same number of channels to output and the same number of the data.
- Only start-trigger with no delay is available for triggering in this parallel analog output.
- Available trigger modes depend on the data transfer mode.

Data Transfer Mode	Trigger Modes Available
Programmed I/O	External trigger External trigger with mask using general purpose digital input pin
Memory	External trigger

- You can specify the master mode only one of the boards in the parallel analog output. The others should be specified as the slave mode.
- In execution order of the DaSyncSampling function, first, you should call this function to each salve board in sequence to place them into the ready state, and then call this function to the master board to start the analog output update in parallel.

Example

```
int nRet;

nRet = DaSyncSampling(2, DA_SLAVE_MODE);
nRet = DaSyncSampling(1, DA_MASTER_MODE);
```

Configure a board whose device number is 2 as a slave board and a board whose device number is 1 as a master board, then start simultaneous analog output update in parallel.

5.1.16 DaStopSampling

The DaStopSampling function stops the analog output update performed as an overlapped operation.

```
int DaStopSampling(
  int           nDevice
);
```

Parameter

nDevice Specifies the device number opened by the DaOpen function.

Return Value

The DaStopSampling function returns AD_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_STOP_SAMPLING

Comment

If the callback routine is set by using the DaSetBoardConfig function, after this function is called, the callback function executes.

Example

```
int nRet;

nRet = DaStopSampling(1);
```

Stop the analog output update of the board whose device number is 1 immediately.

5.1.17 DaGetStatus

The DaGetStatus function retrieves the analog output update status of the board.

```
int DaGetStatus(
  int              nDevice,
  unsigned long*   ulDaSmplStatus,
  unsigned long*   ulDaSmplCount,
  unsigned long*   ulDaAvailCount,
  unsigned long*   ulDaAvailRepeat
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulDaSmplStatus Points to a variable to receive the output status. The variable will contain one of the followings.

Code	Description
DA_STATUS_STOP_SAMPPLING	The analog output update is stopped.
DA_STATUS_WAIT_TRIGGER	The analog output update is waiting for a trigger.
DA_STATUS_NOW_SAMPLING	The analog output update is running.

ulDaSmplCount Points to a variable to receive the number of data that have already been output.

ulDaAvailCount Points to a variable to receive the number of data not to be output.

ulDaAvailRepeat Points to a variable to receive the repetition counts not to be done. (The PCI-3335 and PCI-3337 boards can not retrieve the repetition counts. The value is always 0.)

Return Value

The DaGetStatus function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following code. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NULL_POINTER

Example

```
int nRet;
unsigned long ulDaSmplStatus;
unsigned long ulDaSmplCount;
unsinged long ulDaAvailCount;
unsigend long ulDaAvailRepeat;

nRet = DaGetStatus(1, &ulDaSmplStatus, &ulDaSmplCount,
 &ulDaAvailCount, &ulDaAvailRepeat);
if(!nRet){
   printf("Status: %X\n",ulDaSmplStatus);
   printf("Count: %d\n",ulDaSmplCount);
}
```

Retrieve the analog output update status on the board whose device number is 1.

5.1.18 DaSetOutputMode

The DaSetOutputMode function enables or disables the simultaneous analog output. This function is applicable only to the PCI/PAZ-3329, PCI/PAZ-3338, CTP-3329, and CTP-3338.

```
int DaSetOutputMode(
  int           nDevice,
  unsigned long ulMode
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulMode Specifies the simultaneous analog output enabled or disabled.

Code	Description
DA_SYNC_OUTPUT	Enables the simultaneous analog output.
DA_NORMAL_OUTPUT	Disables the simultaneous analog output. (default setting)

Return Value

The DaSetOutputMode function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_INVALID_PARAMETER

Example

```
int nRet;

nRet = DaSetOutputMode(1, DA_SYNC_OUTPUT);
```

Enable the simultaneous analog output of the board whose device number is 1.

5.1.19 DaGetOutputMode

The DaGetOutputMode function retrieves the configuration of the simultaneous analog output. This function is applicable only to the PCI/PAZ-3329, PCI/PAZ-3338, CTP-3329, and CTP-3338.

```
int DaGetOutputMode(
  int            nDevice,
  unsigned long  ulMode
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulMode Points to a variable to receive the configration of the simultaneous analog output.

Code	Description
DA_SYNC_OUTPUT	The simultaneous analog output is enabled.
DA_NORMAL_OUTPUT	The simultaneous analog output is disabled. (default setting)

Return Value

The DaGetOutputMode function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_NULL_POINTER

Example

```
int nRet;
Unsigned long ulMode;

nRet = DaGetOutputMode(1, &ulMode);
```

Retrieve the configuration of the simultaneous analog outupt of the board whose device number is 1.

5.1.20 DaOutputDA

The DaOutputDA function outputs one-shot analog data on the board.

```
int DaOutputDA(
  int            nDevice,
  unsigned long  ulCh,
  PDASMPLCHREQ   pSmplChReq,
  void*          pData
);
```

Parameters

nDevice	Specifies the device number opened by the DaOpen function.
ulCh	Specifies the number of channels to which data are output. Each channel number is specified in the ulChNo member of the DASMPLCHREQ structure. The settable range is 1 through the maximum number of channels of the board.
pulSmplChReq	Points to the DASMPLCHREQ structure.
pData	Points to the buffer containing data to be output. Please refer to "4.2 Data Format."

Return Value

The DaOutputDA function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_ILLEGAL_PARAMETER
- DA_ERROR_INVALID_PARAMETER
- DA_ERROR_NULL_POINTER

Comment

For example, if you want to output data on the channel 1, channel 3, channel 5, and channel 7, the number of channel is four. Each channel number is stored in the ulChNo member of the DASMPLCHREQ structure.

Example

```
int nRet;
DASMPLCHREQ SmplChReq[4];

SmplChReq[0].ulChNo = 1; SmplChReq[1].ulChNo = 3;
SmplChReq[2].ulChNo = 5; SmplChReq[3].ulChNo = 7;
nRet = DaOutputDA(1, 4, &SmplChReq[0], pData);
```

Output data to channel 1, channel 3, channel 5, and channel 7 on the board whose device number is 1.

5.1.21 DaInputDI

The DaInputDI function reads general purpose digital input pins on the board.

```
int DaInputDI(
  int             nDevice,
  unsigned long*  ulData
);
```

Parameters

nDevice	Specifies the device number opened by the DaOpen function.
ulData	Points to a variable to receive the digital input data. Please refer to "4.2 Data Format."

Return Value

The DaInputDI function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_NULL_POINTER

Comment

The DaInputDI function isn't applicable to the board (PCI/PAZ-3305) that has no general purpose digital input pins.

Example

```
int nRet;
unsigned long ulData;

nRet = DaInputDI( 1, &ulData );
if(!nRet){
printf("Input Data: %X\n",ulData);
}
```

Read the status of general purpose digital input pins on the board whose device number is 1.

5.1.22 DaOutputDO

The DaOutputDO function writes data to general purpose digital output pins on the board.

```
int DaOutputDO(
  int           nDevice,
  unsigned long ulData
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulData Specifies the digital data to be output. Please refer to "4.2 Data Format."

Return Value

The DaOutputDO function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NOT_SUPPORTED
- DA_ERROR_INVALID_PARAMETER

Comment

The DaOutputDO function isn't applicable to the board (PCI/PAZ-3305) that has no general purpose digital output pins.

Example

```
int nRet;

nRet = DaOutputDO(1, 0x03);
```

Write data 03h to the general purpose digital output pins on the board whose device number is 1.

5.1.23 DaSetFifoConfig

The DaSetFifoConfig function configures analog output update conditions of the board at FIFO data transfer mode. This function is applicable only to the PCI-3525.

```
int DaSetFifoConfig(
  int            nDevice,
  PDAFIFOREQ     pDaFifoConfig
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

pDaFifoConfig Points to the DAFIFOREQ structure.

Return Value

The DaSetFifoConfig function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_ILLEGAL_PARAMETER
- DA_ERROR_NULL_POINTER
- DA_ERROR_NOT_SUPPORTED

Comment

The number of analog output channels and repetitions cannot be changed while the analog output is running.

Example

```
int  nRet;
DAFIFOREQ  DaFifoConfig;

DaFifoConfig.ulChCount = 1;
DaFifoConfig.SmplChReq[0].ulChNo = 1;
DaFifoConfig.SmplChReq[0].ulRange = DA_5V;
DaFifoConfig.fSmplFreq = 10000;
DaFifoConfig.ulSmplRepeat = 1;
DaFifoConfig.ulSmplNum = 100;
DaFifoConfig.ulStartTrgCondition = DA_TRG_FREERUN;
DaFifoConfig.ulStopTrgCondition = DA_TRG_SMPLNUM;
DaFifoConfig.ulEClkEdge = DA_DOWN_EDGE;
DaFifoConfig.ulETrgEdge = DA_START_DOWN_EDGE;

nRet = DaOpen(1);
if(nRet) exit(1);

nRet = DaSetFifoConfig( 1, &DaFifoConfig );
```

Configure the analog output update conditions of the board whose device number is 1.

5.1.24 DaGetFifoConfig

The DaGetFifoConfig function retrieves analog output update conditions of the board at FIFO data transfer mode. This function is applicable only to the PCI-3525.

```
int DaGetFifoConfig(
  int           nDevice,
  PDAFIFOREQ    pDaFifoConfig
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

pDaFifoConfig Points to the DAFIFOREQ structure to receive analog output update conditions.

Return Value

The DaGetFifoConfig function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NULL_POINTER
- DA_ERROR_NOT_SUPPORTED

Comment

The default settings of the DAFIFOREQ structure can be retrieved by calling the DaGetFifoConfig function immediately after opening the board.

Example

```
int  nRet;
unsigned long  i;
DAFIFOREQ  DaFifoConfig;

nRet = DaOpen(1);
if(nRet) exit(1);

nRet = DaGetFifoConfig(1, &DaFifoConfig);
if(!nRet){
    if( i=0; i<DaFifoConfig.ulChCount; i++){
       printf("Output channel: %d\n",DaFifoConfig.SmplChReq[i].ulChNo);
    }
}
```

Retrieve the analog output update conditions of the board whose device number is 1.

5.1.25 DaSetInterval

The DaSetInterval function configures the interval timer cycle. This function is applicable only to the PCI-3525.

```
int DaSetInterval(
  int              nDevice,
  unsigned long    ulInterval
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulInterval Specifies the interval timer cycle in the range of 0 through 16777215 in us. If the cycle is 0, the interval timer will be stopped.

Note: “us” means microsecond.

Return Value

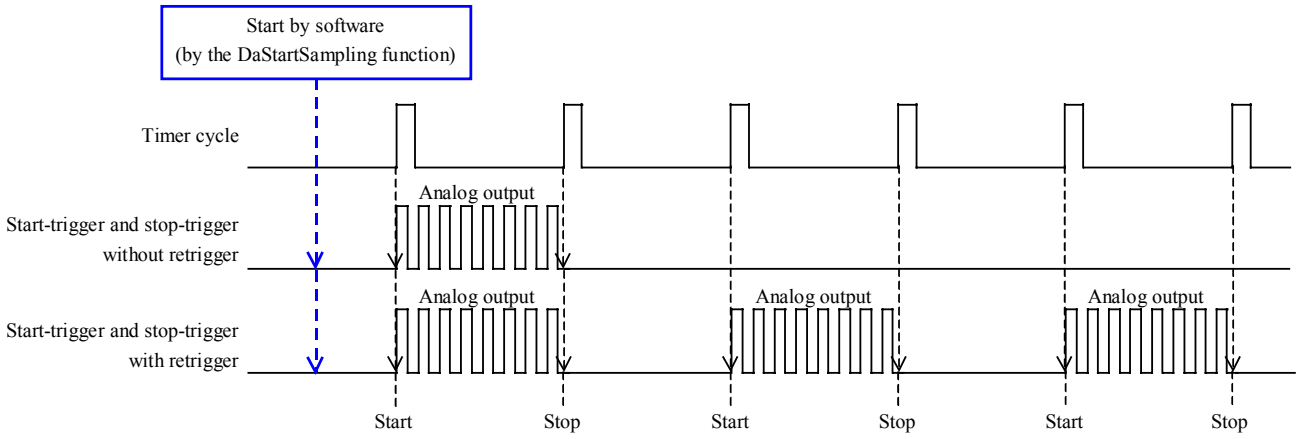
The DaSetInterval function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_INVALID_PARAMETER
- DA_ERROR_NOT_SUPPORTED

Comment

The following figure shows behavior of analog output update when an interval timer is set to both the start-trigger condition and stop-trigger condition.

With the first timer event, analog output update starts. With the second timer event, the analog output update stops. If retrigger capability is set to the conditions, these operations will be performed repeatedly with timer events.



Example

```
int  nRet;

nRet = DaOpen(1);
if(nRet) exit(1);

nRet = DaSetInterval(1, 1000);
```

Configure the interval timer cycle to 1 ms on the board whose device number is 1.

5.1.26 DaGetInterval

The DaGetInterval function retrieves the interval timer cycle. This function is applicable only to the PCI-3525.

```
int DaGetInterval(
  int             nDevice,
  unsigned long   *pulInterval
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

pulInterval Points to a variable to receive the interval timer cycle in the range of 0 through 16777215 in us. If the cycle is 0, the interval timer is stopped.

Note: “us” means microsecond.

Return Value

The DaGetInterval function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_NULL_POINTER
- DA_ERROR_NOT_SUPPORTED

Example

```
int  nRet;
unsigned long ulInterval;

nRet = DaOpen(1);
if(nRet) exit(1);

nRet = DaGetInterval(1, &ulInterval);
if(nRet == DA_ERROR_SUCCESS) printf("Interval = %lu\n", ulInterval);
```

Retrieve the interval timer cycle on the board whose device number is 1.

5.1.27 DaSetFunction

The DaSetFunction function configures the function of the CN3 connector. This function is applicable only to the PCI-3525.

```
int DaSetFunction(
  int               nDevice,
  unsigned long     ulCnNo,
  unsigned long     ulFunction
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulCnNo Specifies the connector number to configure the function. Specify 3.

ulFunction Specifies the function of the connector specified by ulCnNo.

Code	Description
DA_CN_FREE	The connector is not used. (default setting)
DA_CN_EXTRG_IN	External trigger input
DA_CN_EXTRG_OUT	External trigger output
DA_CN_EXCLK_IN	External clock input
DA_CN_EXCLK_OUT	External clock output
DA_CN_EXINT_IN	External interrupt input
DA_CN_ATRG_OUT	Analog trigger output
DA_CN_DI	General purpose digital input
DA_CN_DO	General purpose digital output

Return Value

The DaSetFunction function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_INVALID_PARAMETER
- DA_ERROR_USED_AD
- DA_ERROR_NOT_SUPPORTED

Comments

- The operation of this function depends on the setting of CN3 in the GPH-3100. Refer to the following table to check variable combination of codes before you use this function.

GPH-3300 / GPH-3100	DA_CN_FREE	Other codes
AD_CN_FREE	Available	Available
Other codes	Available	Not available

- When DA_CN_DI is specified for CN3, CN3 is IN1.
- When DA_CN_DO is specified for CN3, CN3 is OUT1.

Example

```
int  nRet;
unsigned long ulCnNo = 3;
unsigned long ulFunction = DA_EXTRG_IN;

nRet = DaOpen(1);
if(nRet) exit(1);

nRet = DaSetFunction(1, ulCnNo, ulFunction);
```

Configure the function of the CN3 connector to an external trigger input on the board whose device number is 1.

5.1.28 DaGetFunction

The DaGetFunction function retrieves the functional configuration of the CN3 connector. This function is applicable only to the PCI-3525.

```
int DaGetFunction(
  int               nDevice,
  unsigned long     ulCnNo,
  unsigned long     *pulFunction
);
```

Parameters

nDevice Specifies the device number opened by the DaOpen function.

ulCnNo Specifies the connector number to retrieve the functinal configuration. Specify 3.

pulFunction Points to a variable to receive the function of the connector specified by ulCnNo.

Code	Description
DA_CN_FREE	The connector is not used. (default setting)
DA_CN_EXTRG_IN	External trigger input
DA_CN_EXTRG_OUT	External trigger output
DA_CN_EXCLK_IN	External clock input
DA_CN_EXCLK_OUT	External clock output
DA_CN_EXINT_IN	External interrupt input
DA_CN_ATRG_OUT	Analog trigger output
DA_CN_DI	General purpose digital input
DA_CN_DO	General purpose digital output

Return Value

The DaGetFunction function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NOT_DEVICE
- DA_ERROR_INVALID_DEVICE_NUMBER
- DA_ERROR_INVALID_PARAMETER
- DA_ERROR_NULL_POINTER
- DA_ERROR_USED_AD
- DA_ERROR_NOT_SUPPORTED

Comments

- The operation of this function depends on the setting of CN3 in the GPH-3100. Refer to the following table to check variable combination of codes before you use this function.

GPH-3300 / GPH-3100	DA_CN_FREE	Other codes
AD_CN_FREE	Available	Available
Other codes	Available	Not available

- The default value is retrieved by calling this function if the functional configurations are not changed in the DaSetFunction.

Example

```
int  nRet;
int  nDevice = 1;
unsigned long ulCnNo = 3;
unsigned long ulFunction;

nRet = DaOpen(nDevice);
if(nRet) exit(1);

nRet = DaGetFunction(nDevice, ulCnNo, &ulFunction);
if(!nRet) printf("CN%d , FUNCTION:%lx\n", ulCnNo, ulFunction);
```

Retrieve the functional configuration of the CN3 connector on the board whose device number is 1.

5.1.29 DaDataConv

The DaDataConv function converts forms of the analog data. Averaging and interpolation can be done with the conversion. You can supply the user-defined function to perform user specific conversion.

```
int DaDataConv(
  unsigned long   ulSrcFormCode,
  void*           pSrcData,
  unsigned long   ulSrcSmplDataNum,
  PDASMPLREQ      pSrcSmplReq,
  unsigned long   ulDestFormCode,
  void*           pDestData,
  unsigned long*  pulDestSmplDataNum,
  PDASMPLREQ      pDestSmplReq,
  unsigned long   ulEffect,
  unsigned long   ulCount,
  CONVPROC        pfnConv
);
```

Parameters

ulSrcFormCode Specifies an original data form stored in the buffer pointed by pSrcData.

Code	Description
DA_DATA_PHYSICAL	Physical value (voltage [V] or current [mA])
DA_DATA_BIN8	8-bit binary
DA_DATA_BIN12	12-bit binary
DA_DATA_BIN16	16-bit binary
DA_DATA_BIN24	24-bit binary

The binary data means that the data can be input from or output to the board directly.

pSrcData Points to the source data to be converted.

ulSrcSmplDataNum Specifies the number of source data to be converted.

pSrcSmplReq Points to the DASMPLREQ structure containing the analog output conditions of the source data.

ulDestFormCode Specifies an original data form stored in the buffer pointed by pDestData.

Code	Description
DA_DATA_PHYSICAL	Physical value (voltage [V] or current [mA])
DA_DATA_BIN8	8-bit binary
DA_DATA_BIN12	12-bit binary
DA_DATA_BIN16	16-bit binary
DA_DATA_BIN24	24-bit binary

The binary data means that the data can be input from or output to the board directly.

pDestData Points to the buffer to receive data converted.

pulDestSmplDataNum Points to a variable to receive the number of data converted.

pDestSmplReq Points to the DASMPLREQ structure to receive the analog output condition of converted data.

ulEffect Specifies the additional data processing.

Code	Description
0	No averaging and no interpolation.
DA_CONV_SMOOTH	Converts the data with interpolation.
DA_CONV_AVERAGE1	Converts the data with the simple averaging.
DA_CONV_AVERAGE2	Converts the data with the shifted averaging.

ulCount Specifies the number of original data to average or interpolate. If ulEffect is set to 0, ulCount is ignored.

pfnConv Points to the user-supplied function to achieve arbitrary data processing. If you don't use this capability, specify NULL to pfnConv.

Return Value

The DaDataConv function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NULL_POINTER
- DA_ERROR_INVALID_DATA_FORMAT
- DA_ERROR_INVALID_AVERAGE_OR_SMOOTHING
- DA_ERROR_INVALID_SOURCE_DATA

Comment

If averaging or interpolation is applied to the data processing, the analog output conditions of the converted data are changed from the original depending on the additional processing conditions.

Example

```
int nRet;

nRet = DaDataConv( DA_DATA_BIN12, &pSrcBuffer, &pSrcSmplDataNum,
 &pSrcSmplReq, DA_DATA_BIN16, &pDestBuffer, &puDestSmplDataNum,
 &pDestSmplReq, 0, 0, NULL);
```

The conversion is done under the conditions as follows:

- Source data format: 12-bit binary
- Analog output conditions for source data: Specified by the DASMPLREQ structure.
- Converted data format: 16-bit binary
- Analog output conditions for converted data: Stored to the DASMPLREQ structure.
- Additional data processing: None
- User function: None

5.1.30 DaWriteFile

The DaWriteFile function writes data to the file from the buffer. Binary and CSV formats are supported.

```
int DaWriteFile(
  char*          pszPathName,
  void*          pSmplData,
  unsigned long  ulFormCode,
  unsigned long  ulSmplNum,
  unsigned long  ulChCount
);
```

Parameters

pszPathName Specifies the path to the data file.

pSmplData Points to the buffer containing analog data to be saved.

ulFormCode Specifies the data format.

Code	Description
DA_DATA_PHYSICAL	Physical value (voltage [V] or current [mA])
DA_DATA_BIN8	8-bit binary
DA_DATA_BIN12	12-bit binary
DA_DATA_BIN16	16-bit binary
DA_DATA_BIN24	24-bit binary

ulSmplNum Specifies the number of analog data.

ulChCount Specifies the number of channels.

Return Value

The DaWriteFile function returns DA_ERROR_SUCCESS if the process is successfully completed. Otherwise, this function returns the following codes. Please refer to the error codes.

- DA_ERROR_NULL_POINTER
- DA_ERROR_FILE_OPEN
- DA_ERROR_FILE_CLOSE
- DA_ERROR_FILE_WRITE
- DA_ERROR_INVALID_DATA_FORMAT

Comment

The data are written into the file in the same format in the buffer, the binary data are written into the binary format file, the physical data are written into the CSV format file.

Example

```
int nRet;
char *pszPathName = "DATA.CSV";

nRet = DaWriteFile( pszPathName, pSmplData, DA_DATA_PHYSICAL, 1024, 1 );
```

Write 1024 physical data per channel into the DATA.CSV file from the buffer (pSmplData).

5.1.31 fnConv

The fnConv function is a placeholder for a callback routine used in the DaDataConv function. This function is called when each data is converted.

```
CONVPROC fnConv(
  int           nCh,
  unsigned long ulCount,
  void*         pData
);
```

Parameters

nCh	Contains the channel number of the data to which pData points.
ulCount	Contains an index of the data pointed by pData in the buffer.
pData	Points to the data to be processed in this function. After processing, restore the processed data into the area pointed by pData.

Return Value

The fnConv function has no return value.

5.1.32 CallbackProc

This CallbackProc function is a placeholder for a callback routine. This function is called when the analog output is completed. Supply a pointer to your function for the lpCallbackProc parameter in the DaSetBoardConfig function.

```
LPDACALLBACK CallbackProc(
  int          nReserved
);
```

Parameter

nReserved Reserved.

Return Value

This function has no return value.

5.2 Structures

5.2.1 DASMPLREQ Structure

The DASAMPLREQ structure contains analog output conditions. This structure is used by the DaDataConv and DaSetSamplingConfig functions.

```
typedef struct {
  unsigned long    ulChCount;
  DASMPLCHREQ      SmplChReq[256];
  unsigned long    ulSamplingMode;
  float            fSmplFreq;
  unsigned long    ulSmplRepeat;
  unsigned long    ulTrigMode;
  unsigned long    ulTrigPoint;
  unsigned long    ulTrigDelay;
  unsigned long    ulEClkEdge;
  unsigned long    ulTrigEdge;
  unsigned long    ulTrigDI;
} DASMPLREQ, *PDASMPLREQ
```

Member	Description
ulChCount	Specifies the number of channels to output data. It is in the range of 1 through the maximum number of channels that the board provides. The default seting value is 1. Specifies the channel numbers in the DASMPLCHREQ structure.
SmplChReq	Specifies the DASMPLCHREQ structure containing the analog output conditions for each channel.
ulSamplingMode	Specifies the data transfer mode. Available modes depend on the board.

Code	Description
DA_IO_SAMPLING	Programmed I/O
DA_FIFO_SAMPLING	FIFO
DA_MEM_SAMPLING	Memory

fSmplFreq	Specifies the analog output update rate in Hz. You can specify this from 0.01f to the maximum output update rate that the board supports. To use the external clock, please specify 0.0f to this member. The default setting is depending on the board. The default values are retrieved by calling the DaGetSamplingConfig function after the board is opened.

`ulSmplRepeat` Specify the repetitions of analog output from 1 through 65535. When you specify 0, the driver software repeatedly updates analog outputs until the DaStopSampling function is called.

The default setting value is 1.

`ulTrigMode` Specifies the trigger mode. One of the following codes must be set exclusively.

Code	Description
DA_FREERUN	No trigger (default setting)
DA_EXTTRG	External trigger
DA_EXTTRG_DI	External trigger with mask using general purpose digital input pin

`ulTrigPoint` Specifies the trigger timing. One of the following codes must be set exclusively.

Code	Description
DA_TRIG_START	Start-trigger (default setting)
DA_TRIG_STOP	Stop-trigger
DA_TRIG_START_STOP	Start/stop-trigger

`ulTrigDelay` Specifies the number of analog output data for post-trigger. This member is available when the trigger mode except DA_FREERUN is set and the trigger timing except DA_TRIG_START_STOP is set.

Number of analog output data for post-trigger: 1 through 1073741824
No trigger delay: 0

The default seting value is 0.

`ulEClkEdge` Specifies the edge polarity of the external clock signal. This member is available when the fSmplFreq is 0.0f.

Code	Description
DA_DOWN_EDGE	Falling edge (default setting)
DA_UP_EDGE	Rising edge

`ulTrigEdge` Specifies the polarity of the external trigger. This member is available when the trigger mode is the external trigger or the external trigger with mask using a general purpose digital input pin.

Code	Description
DA_DOWN_EDGE	Falling edge (default setting)
DA_UP_EDGE	Rising edge

`ulTrigDI` Selects a general purpose digital input pin to be used with the trigger conditions. While the status of the digital input pin is low, the assertion of the external trigger is valid. The number of digital input pins available for the mask setting depends on the board specifications. This member is available when the trigger mode is an external trigger with mask using a digital input pin. The format of ulTrigDI is the same as digital input data. Please refer to "4.2 Data Format."

The default setting value is 0.

5.2.2 DASMPLCHREQ Structure

The DASMPLCHREQ structure contains the channel-specific analog output conditions for each channel. This structure is used for the member of the DASMPLREQ structure and the DaOutputDA function.

```
typedef struct {
  unsigned long    ulChNo;
  unsigned long    ulRange;
} DASMPLCHREQ, *PDASMPLCHREQ
```

Member	Description
ulChNo	Specifies the channel number to output data. The range is from 1 through the maximum number of channels that the board provides.
ulRange	Specifies the output range of the channel specified by ulChNo. Please select one of the following codes.

Code	Description
DA_0_1V	Voltage range: 0 V to +1 V
DA_0_2P5V	Voltage range: 0 V to +2.5 V
DA_0_5V	Voltage range: 0 V to +5 V
DA_0_10V	Voltage range: 0 V to +10 V
DA_1_5V	Voltage range: 1 V to +5 V
DA_0_20mA	Current range: 0 mA to +20 mA
DA_4_20mA	Current range: +4 mA to +20 mA
DA_1V	Voltage range: +/-1 V
DA_2P5V	Voltage range: +/-2.5 V
DA_5V	Voltage range: +/-5 V
DA_10V	Voltage range: +/-10 V

5.2.3 DABOARDSPEC Structure

The DABOARDSPEC structure contains the specifications of the board. This structure is used for the DaGetDeviceInfo function.

```
typedef struct {
  unsigned long    ulBoardType;
  unsigned long    ulBoardID;
  unsigned long    ulSamplingMode;
  unsigned long    ulChCount;
  unsigned long    ulResolution;
  unsigned long    ulRange;
  unsigned long    ulIsolation;
  unsigned long    ulDi;
  unsigned long    ulDo;
} DABOARDSPEC, *PDABOARDSPEC;
```

Member — **Description**

`ulBoardType` — Receives the board model.
Example) If you use the PCI/PAZ-3329, this member will contain 3329 in decimal.
If you use the CTP-3346, this member will contain 3346 in decimal.

`ulBoardID` — Receives the board ID (RSW1 value of the board).

`ulSamplingMode` — Receives the data transfer mode that the board supports.

Bit	Data Transfer Mode
0	Programmed I/O
1	FIFO
2	Memory
3 through 31	Reserved

0: Not supported
1: Supported

`ulChCount` — Receives the number of channels.

`ulResolution` — Receives the resolution of the board.
Example) If you use a 12-bit analog output board, this member will contain 12.

`ulRange` Receives the output ranges that the board supports.

Bit	Description
0	Voltage range: 0 V to +1 V
1	Voltage range: 0 V to +2.5 V
2	Voltage range: 0 V to +5 V
3	Voltage range: 0 V to +10 V
4	Voltage range: 1 V to +5 V
5 through 11	Reserved
12	Current range: 0 mA to +20 mA
13	Current range: +4 mA to +20 mA
14, 15	Reserved
16	Voltage range: +/-1 V
17	Voltage range: +/-2.5 V
18	Voltage range: +/-5 V
19	Voltage range: +/-10 V
20 through 31	Reserved

0: Not supported

1: Supported

`ulIsolation` Receives the isolation capability.

Code	Description
DA_ISOLATION	Isolated
DA_NOT_ISOLATION	Not isolated

`ulDi` Receives the number of the general purpose digital input pins on the board.

`ulDo` Receives the number of the general purpose digital output pins on the board.

5.2.4 DAMODEREQ Structure

The DAMODEREQ structure contains board specific parameters used in the DaSetMode and DaGetMode functions.

```
typedef struct {
  DAMODECHREQ      ModeChReq[2];
  unsigned long    ulSyntheOut;
  unsigned long    ulPulseMode;
  unsigned long    ulInterval;
  float            fIntervalCycle;
  unsigned long    ulCounterClear;
  unsigned long    ulDaLatch;
  unsigned long    ulSamplingClock;
  unsigned long    ulExControl;
  unsigned long    ulExClock;
} DAMODEREQ, *PDAMODEREQ
```

Member — **Description**

`ModeChReq`

Specifies the channel-specific output range configurations (DAMODECHREQ structure). First element and second element of the array correspond channel 1 and channel 2, respectively. You have to configure the condition for 2 channels.

`ulSyntheOut`

Specifies the waveform generation mode.

Code	Description
DA_MODE_CUT	Time-based waveform generation (default setting)
DA_MODE_SYNTHE	Frequency-based waveform generation

`ulPulseMode`

Specifies the multiplier for the frequency-based waveform generation. It must be one of the power of two less than or equal to 524288.

The default setting value is 1.

`ulInterval`

Specifies whether the wait state is inserted or not in the repeat output mode.

Code	Description
DA_REPEAT_NONINTERVAL	Repeat without the wait state (default setting)
DA_REPEAT_INTERVAL	Repeat with the wait state

`fIntervalCycle`

Specifies the frame frequency in the repeat output mode. You can specify it from 0.01 to 2500000 in Hz.
The default settoing value is 1.0 Hz.

`ulCounterClear` Specifies the analog output counter status when the analog output update starts.

Code	Description
DA_COUNTER_CLEAR	Cleared (default setting)
DA_COUNTER_NONCLEAR	Not cleared

`ulDaLatch` Specifies whether the output voltages are hold (DA latch not cleared) or set to the lowest voltage of the range (DA latch cleared) when the analog output is completed.

Code	Description
DA_LATCH_CLEAR	The voltage is set to the lowest voltage of the range.
DA_COUNTER_NONCLEAR	The voltage is held.

`ulSamplingClock` Specifies the analog output update pacer clock source. The internal programmable timer enables the output update rate up to 2.5 MHz in variable. Fixed 5 MHz clock source is also available.

Code	Description
DA_CLOCK_TIMER	The update pacer clock source is the internal programmable timer. The frequency is 2.5 MHz at the default setting.
DA_CLOCK_FIXED	The update pacer clock source is the fixed 5 MHz clock.

`ulExControl` For the PCI/PAZ-3305:
Specifies the configurations of the connector CN3.

Code	Description
DA_EXTRG_IN	External trigger input (default setting)
DA_EXTRG_OUT	External trigger output

For the PCI-3335 and PCI-3337:
Specifies the mode of the EXTRG OUT pin.

Code	Description
DA_EXTRG_IN	Disables the external trigger output
DA_EXTRG_OUT	Enables the external trigger output (default setting)

`ulExClock`

For the PCI/PAZ-3305:

Specifies the configurations of the connector CN4.

Code	Description
DA_EXCLK_IN	External clock input (default setting)
DA_EXCLK_OUT	External clock output

For the PCI/PAZ-3310, PCI-3335, PCI/PAZ-3336, PCI-3337, and PCI/PAZ-3340:

Specifies the mode of the EXCLK OUT pin.

Code	Description
DA_EXCLK_IN	Disables the external clock output
DA_EXCLK_OUT	Enables the external clock output (default setting)

5.2.5 DAMODECHREQ Structure

The DAMODECHREQ structure contains the channel-specific analog output range configurations.
This structure is one of members of the DAMODEREQ structure and is used for the DaSetMode function.

```
typedef struct {
  unsigned long    ulRange;
  float            fVolt;
  unsigned long    ulFilter;
} DAMODECHREQ, *PDAMODECHREQ
```

Member — **Description**

`ulRange` — Specifies the analog output range.

Code	Description
DA_RANGE_UNIPOLAR	Unipolar range (default setting)
DA_RANGE_BIPOLAR	Bipolar range

`fVolt` — Specifies an absolute value of the maximum voltage of the range specified by the ulRange member in the range of 1.024 V to 10.0 V.

The voltage is 5.0 V at the default settings for unipolar range: 0 V to +5 V and for bipolar range: -5 V to +5 V.

`ulFilter` — Specifies the low pass filter to reduce the glitches appeared on the output waveforms.

Code	Description
DA_FILTER_OFF	Not used (default setting)
DA_FILTER_ON	Used

5.2.6 DAFIFOREQ Structure

The DAFIFOREQ structure contains analog output update conditions for the FIFO data transfer mode. This structure is used by the DaSetFifoConfig function.

```
typedef struct {
  unsigned long   ulChCount;
  DASMPLCHREQ     SmplChReq[256];
  float           fSmplFreq;
  unsigned long   ulSmplRepeat;
  unsigned long   ulSmplNum;
  unsigned long   ulStartTrigCondition;
  unsigned long   ulStopTrigCondition;
  unsigned long   ulEClkEdge;
  unsigned long   ulTrigEdge;
} DAFIFOREQ, *PDAFIFOREQ
```

Member	Description
ulChCount	Specifies the number of channels to output data. It is in the range of 1 through the maximum number of channels that the board provides. The default setting value is 1. Specifies the channel numbers in the DASMPLCHREQ structure.
SmplChReq	Specifies the DASMPLCHREQ structure containing the analog output conditions for each board.
fSmplFreq	Specifies the analog output update rate in Hz. You can specify this from 0.01f to the maximum output update rate that the board supports. To use the external clock, please specify 0.0f to this member. The default setting is depending on the board. The default values are retrieved by calling the DaGetSamplingConfig function after the board is opened.
ulSmplRepeat	Specify the repetitions of analog output from 1 through 65535. When you specify 0, the driver software repeatedly updates analog outputs until the DaStopSampling function is called or the stop condition is satisfied. The default setting value is 1.

`ulSmplNum`

Specifies the event interval according to the count of output update from 1 through 16777215. When DA_TRG_SMPLNUM is specified to ulStopTrigCondition, analog output update will be stopped when the event is occurred.

The default setting value is 1.

`UlStartTrigCondition` *1

Specifies start-trigger condition of analog output. One of the following codes must be set exclusively.

Code	Description
DA_TRG_FREERUN	No trigger (default setting)
DA_TRG_EXTTRG	External trigger
DA_TRG_ATRG	Analog trigger
DA_TRG_SIGTIMER	Interval timer
DA_TRG_AD_START	AD start
DA_TRG_AD_STOP	AD stop
DA_TRG_AD_PRETRG	AD pre-trigger
DA_TRG_AD_POSTTRG	AD post-trigger

`ulStopTrigCondition` *1

Specifies stop-trigger condition of analog output. One of the following codes must be set exclusively.

Code	Description
DA_TRG_FREERUN	No trigger
DA_TRG_EXTTRG	External trigger
DA_TRG_ATRG	Analog trigger
DA_TRG_SIGTIMER	Interval timer
DA_TRG_AD_START	AD start
DA_TRG_AD_STOP	AD stop
DA_TRG_AD_PRETRG	AD pre-trigger
DA_TRG_AD_POSTTRG	AD post-trigger
DA_TRG_SMPLNUM	The specified number of data are output.
DA_TRG_FIFO_EMPTY	FIFO empty (default setting) *2

The following codes are ORed with the stop-trigger condition.

Code	Description
DA_RETRG	Retrigger *3
DA_FIFO_RESET	Resets FIFO. *4

`ulEClkEdge` Specifies the edge polarity of the external clock signal. This member is available when the fSmplFreq is 0.0f.

Code	Description
DA_DOWN_EDGE	Falling edge (default setting)
DA_UP_EDGE	Rising edge

`ulTrigEdge`[*5] Specifies an edge polarity of each start-trigger and/or stop-trigger when an external trigger is used. Use an OR operator when a code is required for each.

<Start-trigger>

Code	Description
DA_START_DOWN_EDGE	Falling edge (default setting)
DA_START_UP_EDGE	Rising edge

<Stop-trigger>

Code	Description
DA_STOP_DOWN_EDGE	Falling edge (default setting)
DA_STOP_UP_EDGE	Rising edge

Notes:

- [*1] Start/Stop-trigger condition:

  If the same condition is set to both the start-trigger condition and stop-trigger condition, the operation is toggeled. When the first time the condition is satisfied, analog output update starts. At the next time the condition is satisfied, the output update stops. With the retrigger function, output start and stop will be repeated alternately.

- [*2] FIFO empty:

  If DA_TRG_FIFO_EMPTY is set to the stop-trigger condition, analog output update will stop when the output FIFO buffer is empty. Analog output update will not start even if you set the new data to the FIFO buffer.

  If both DA_TRG_FIFO_EMPTY and DA_RETRG are set to the stop-trigger condition, you must set the new data when analog output update is finished. If not so, analog output update will not restart by retrigger capability.

  If a condition except DA_TRG_FIFO_EMPTY is specified, the last data will be output until the new data is set.

- [*3] Retrigger:

  This capability makes analog output update restart when the condition; the start-trigger condition is satisfied after analog output is finished, is satisfied.

  The DA_RETRG cannot be set with DA_TRG_FREERUN.

- [*4] FIFO reset:

  If DA_FIFO_RESET is set, the data will be output from the head of the FIFO at the next analog output update. When this code is not set, the rest data of the previous analog output will be output.

- [*5] Two or more codes setting:

  To set two or more codes, use OR operators.

  Example)

```
DAFIFOREQ FifoConfig;

FifoConfig.ulStopTrigCondition
      = DA_TRG_SIGTIMER | DA_RETRG | DA_FIFO_RESET;
FifoConfig.ulTrigEdge
      = DA_START_DOWN_EDGE | DA_STOP_UP_EDGE;
```

5.3 Return Values

Error Code	Value	Description	Comments/Solutions
DA_ERROR_SUCCESS	0	The process was successfully completed.	-
DA_ERROR_NOT_DEVICE	0xC0000001	The specified driver cannot be called.	The specified device is not found. Make sure that the board and the device driver correctly installed in your computer.
DA_ERROR_NOT_OPEN	0xC0000002	The specified driver cannot be opened.	Errors occurred while the system opens the device.
DA_ERROR_INVALID_DEVICE_NUMBER	0xC0000003	The device number is invalid.	Use the device number obtained by the device number setting program.
DA_ERROR_ALREADY_OPEN	0xC0000004	The specified device cannot be opened because it has already been opened by another process.	The device used by another process cannot be accessed.
DA_ERROR_NOT_SUPPORTED	0xC0000009	The specified function is not supported.	The function is not available because the board does not support.
DA_ERROR_NOW_SAMPLING	0xC0001001	The analog output is running now.	The specified analog output has already been called. The specified function is not available while the analog output is running.
DA_ERROR_STOP_SAMPLING	0xC0001002	The analog output is stopped.	The specified analog output is not available while the analog output has been stopped.
DA_ERROR_START_SAMPLING	0xC0001003	Failed to start the analog output.	There is no analog output data in the output buffer.
DA_ERROR_INVALID_PARAMETER	0xC0001021	The specified parameters are invalid.	Specify correct values.
DA_ERROR_ILLEGAL_PARAMETER	0xC0001022	The specified analog output settings are invalid.	Invalid analog output conditions are specified.
DA_ERROR_NULL_POINTER	0xC0001023	A NULL pointer is specified.	A NULL pointer is specified for source data in the data conversion function. The pointer to the buffer that receives the converted data is NULL.
DA_ERROR_SET_DATA	0xC0001024	The time-out interval elapsed while the analog output is running.	The analog output data couldn't be obtained. The buffer is cleared. The buffer is empty so no data is returned.

(Continued)

Error Code	Value	Description	Comments/Solutions
DA_ERROR_USED_AD	0xC0001025	The AD driver is using the specified function now.	Release the functional configuration that the AD driver is using. Then, call the function again.
DA_ERROR_FILE_OPEN	0xC0001041	Failed to open the file.	The specified file doesn't exist.
DA_ERROR_FILE_CLOSE	0xC0001042	Failed to close the file.	Errors occurred while the file is accessed.
DA_ERROR_FILE_READ	0xC0001043	Failed to read the file.	Errors occurred while the file is accessed.
DA_ERROR_FILE_WRITE	0xC0001044	Failed to write the file.	Errors occurred while the file is accessed.
DA_ERROR_INVALID_DATA_FORMAT	0xC0001061	The specified data format is invalid.	Use valid data formats.
DA_ERROR_INVALID_AVERAGE_OR_SMOOTHING	0xC0001062	The specified averaging or interpolations are invalid.	The number of averaging or interpolations is invalid.
DA_ERROE_INVALID_SOURCE_DATA	0xC0001003	Data specified as source is invalid.	Make sure that specified address of source data is correct.
DA_ERROR_NOT_ALLOCATE_MEMORY	0xC0001081	Not enough memory.	Not enough memory is available to process.

5.4 Kylix

In this document, all examples of programs are written in C. This section contains helpful information for Kylix programmers.

Note: Kylix does not support SH.

5.4.1 Function Definitions

C

```
long DaSetBoardConfig(
  int             nDevice,
  unsigned        ulSmplBufferSize,
  long*
  void*           pReserved,
  PLPDACALLBACK   pCallBackProc,
  int             nReserved
);
```

Kylix

```
1)function DaSetBoardConfig(
                  2)nDevice:  Integer3);
  4)var ulSmplBufferSize:  Cardinal;

                pReserved:  pointer;
            pCallBackProc:  PLPDACALLBACK;
                nReserved:  Integer;
): 5)Longint; 6)cdecl; external
'gpg3300.so';
```

1) In Kylix, a function-module that has a return value uses the `function` reserved word.
   A function-module that has no return value uses the `procedure` reserved word.
2) In Kylix, a variable is written in front of the data type of that.
3) In Kylix, data type are written in the different way of C.
   Example)

C	Kylix
int	Integer
long	Longint
unsigned long	Cardinal
void*	pointer

4) When you write a variable passed by reference, write `var` in front of the variable.
5) In Kylix, data type is written at the end of the variable argument list.
6) To call the library function, write `cdecl;external 'library name';`.
   To call the callback routine, you don't need to write `external 'library name';`.

5.4.2 Structure

C

```
typedef struct{
  unsigned long   ulChNo;
  unsigned long   ulRange;
DASMPLCHREQ, *PDASMPLCHREQ;
```

Kylix

```
type
1)DASMPLCHREQ = record
              2)ulChNo;  Cardinal,
               ulRange;  Cardinal,
3)end;
```

1) In Kylix, a structure is called a record, and write `'structure name' = record`.
2) In Kylix, a variable is written in front of the data type of that.
3) In Kylix, write `end;` at the end of the record.

5.4.3 Example

The following programs show how to write a callback routine.

C

```
void CALLBACK CallBackProc(int nReserved);

void main()
{
    unsigned long* ulSmplBufferSize;

    DaOpen(1);
    ulSmplBufferSize = 2048;
    DaSetBoardConfig(1, ulSmplBufferSize, NULL, CallBackProc, 0);
    :
    :
}

void CallBackProc(int nReserved)
{
    // Write processing of the callback routine.
}
```

Kylix

```
1)var
    procedure CallBackProc(nReserved:Integer);cdecl;

procedure TForm1.FormCreate(Sender: TObject);
var
    ulSmplBufferSize: Cardinal;
2)begin

    DaOpen(1);
    ulSmplBufferSize := 2048;
    DaSetBoardConfig(1, ulSmplBufferSize, NULL, CallBackProc, 0);

end;

procedure CallBackProc (nReserved:Integer);cdecl;
begin
    // Write processing of the callback routine.
end;
```

1) Declare a variable or function after the `var` reserved word.
2) Write codes between `begin` and `end;`.
3) In Kylix, the assignment operator is :=.
4) In Kylix, the address operator is @.
5) In Kylix, a leading $ means hexadecimal.

5.5 Test Driver

The GPH-3300 has the test driver capable of checking the functions of the GPH-3300 without using the board. To use the test driver, link `libgpg3300t.so` instead of `libgpg3300.so`.

The following shows the example to compile the test.c program that uses the test driver.

```
#gcc -o test test.c -lpthread -lgpg3300t
```

Each function checks whether parameters are correctly specified or not. As error codes, refer to Return Value for details.

Function	Description
DaOpen	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaClose	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaCloseEx	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetDeviceInfo	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetBoardConfig	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetBoardConfig	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetSamplingConfig	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetSamplingConfig	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetMode	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetMode	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetSamplingData	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaClearSamplingData	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaStartSampling	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaStartFileSampling	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSyncSampling	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaStopSampling	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetStatus	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetOutputMode	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetOutputMode	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaOutputDA	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaInputDI	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaOutputDO	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetFifoConfig	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetFifoConfig	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetInterval	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetInterval	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaSetFunction	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaGetFunction	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaDataConv	Returns DA_ERROR_SUCCESS if the process was successfully completed.
DaWriteFile	Returns DA_ERROR_SUCCESS if the process was successfully completed.
fnConv	The process was successfully completed.
CallBackProc	The process was successfully completed.

Chapter 6 Sample Programs

Executable files of the sample programs are not included with this product. Please make your executable files before you use the sample programs.

The sample program sources and makefiles are located in the /usr/src/interface/gph3300/i386/linux/samples directory.

6.1 sampledata.c

This sample program performes the analog output update as a non-overlapped operation.

1. Specify the device number to control.
2. Select either of sine waveform or square waveform.
3. Analog output data of 200 samples will be updated. When 0 is specified as the number of repeat, analog output update will be continuously repeated.

The following table shows the analog output update conditions. Configure them according to your board.

Parameter	Setting
Number of channels	2
Data transfer mode	Default value of the board
Analog output update rate	Default value of the board
Channel and range	Channel 1, +/-5 V
Channel and range	Channel 2, +/-5 V

6.2 async.c

This sample program performes the analog output update as an overlapped operation.

1. Specify the device number to control.
2. Select either of sine waveform or square waveform.
3. Analog output data will be continuously updated.
4. To stop output, press the q key.
5. When the analog output update is finished, an ending message will be displayed.

The following table shows the analog output update conditions. Configure them according to your board.

Parameter	Setting
Number of channels	2
Data transfer mode	Default value of the board
Analog output update rate	Default value of the board
Channel and range	Channel 1, +/-5 V
Channel and range	Channel 2, +/-5 V

6.3 outputda.c

This sample program performes one analog output update.

1. Specify the device number to control.
2. Specify the output voltage, and one analog output data will be updated.
3. An operation as step 2 will be continued until the output voltage greater than the full scale voltage is specified.

The following table shows the analog output update conditions. Configure them according to your board.

Parameter	Setting
Number of channels	1
Channel and range	Channel 1, +/-5 V

6.4 file.c

This sample program performs the analog output update from the specified file.

1. Write 1024 binary data to the test.dat file by using the DaWriteFile function.
2. Specify the device number to control.
3. Analog output data from the test.dat file will be updated as a non-overlapped operation.

The following table shows the analog output update conditions. Configure them according to your board.

Parameter	Setting
Number of channels	1
Data transfer mode	Default value of the board
Number of repeat	100
Analog output update rate	1000 Hz
Channel and range	Channel 1, +/-5 V

6.5 fifosampling.c

This sample program performes the analog output update as a non-overlapped operation at the FIFO data transfer mode. This program is applicable only to the PCI-3525.

The following table shows the analog output update conditions. Configure them according to your board.

Parameter	Setting
Number of channels	1
Analog output update rate	10 kHz
Channel and range	Channel 1, +/-5 V

6.6 adasync.c

This sample program synchronously starts the analog output update and sampling on the PCI-3525 board.

1. Spedifiy the device number to control.
2. Configure the analog output update condition to start the analog output synchronizing with the sampling by the DaStartSampling function.
3. The sampling stop when it reaches the specified number. The analog output update stop when the sampling completed.

The following tables show the sampling and analog output update conditions. Configure them according to your boards.

<AD>

Parameter	Setting
Number of channels	1
Sampling rate	1 MHz
Channel and range	Channel 1, +/-5 V
Channel and range	Channel 2, +/-5 V
Start-trigger condition	DA start
Stop-trigger condition	The specified number of data is sampled.

<DA>

Parameter	Setting
Number of channels	1
Analog output update rate	10 kHz
Channel and range	Channel 1, +/-5 V
Start-trigger condition	Software start
Stop-trigger condition	AD stop

6.7 Sample Programs for Kylix

The following sample programs are provided for Kylix.

Sample Program	Description
outputda_k.dpr	Kylix version of the outputda.c sample program
sampledata_k.dpr	Kylix version of the sampledata.c sample program

Compile and run them in the kylix directory as follows.

```
#cd /usr/src/interface/gph3300/i386/linux/samples/kylix
#make
#./sample
```

Chapter 7 Utility Program

7.1 DA Calibration Program

We ship the board after it was fully calibrated at 25 degrees centigrade (77 degrees Fahrenheit). Adjustments and calibration may be necessary under the following conditions.

- Ambient temperature changes
- Output configuration changes
- To optimize measurement accuracy

7.1.1 Required Items for the Calibration Program

- Interface analog output board
- Interface analog output board calibration program
- Terminal block which is appropriate for the board
- Multimeter
  Use 5 or more digit digital multimeter.
- Cables
  Use a shielded cable less than or equal to 50 cm.

7.1.2 Starting the Calibration Program

Change the current directory to `interface/gph3300/i386/bin/` under the target directory at the installation, and run `./cdaadjust`, then the **DA Calibration Program** will start.

This calibration program is necessary for the following boards.

PCI/PAZ-3176	PCI/PAZ-3310	PCI-3335	PCI/PAZ-3336	PCI-3337
PCI/PAZ-3340	PCI-3347	CTP-3340A	CTP-3340B	CTP-3340C
CTP-3340D	CTP-3347			

7.1.3 Selecting the Board

1. Enter the device number. The device number should be set by the device number setting utility (DPG-0101).

```
**********************************************
DA Caribration Program
----------------------------------------------
Version: 1.01-02
----------------------------------------------
Copyright 2000, 2002 Interface Corporation.
              All rights reserved.
**********************************************

Enter the device number:
```

2. The board model and RSW1 value will be displayed.
   If you select the board that is unnessesary to calibrate, the messages "**This board is calibration free. Program is terminated.**" will be displayed, and the program will be terminated.

3. To start the calibration of the selected board, press the y key. If not so, press the n key.

```
===== Board Information========
   Device No.: 1
   Board Type: PCI/PAZ-3310
   RSW1: 0h
==============================

OK? (y/n): y
```

7.1.4 Selecting the Calibration Parameters

1. Configure the calibration parameters and enter the corresponding numbers. The following table shows the calibration parameters.

Parameter	Description
Calibration Channel	Selects the channel.
Range	Selects the range.
Calibration Item	Selects an item.

```
===== Calibration Channel =====
 1 through 4
==============================

 Enter the channel number: 1

===== Range ===============
1: Unipolar: 0 V to +5 V
2: Unipolar: 0 V to +10 V
3: Bipolar: -5 V to +5 V
4: Bipolar: -10 V to +10 V
===========================

Enter the output range: 3
```

2. The selected parameters will be displayed. Please check the selections are correct. Press the y key to start calibration under the conditions. If not so, press the n key.

```
========================================
===== Caribration channel: 1
===== Caribration range: -5 V to +5 V
========================================

OK? (y/n): y
```

3. If you press the n key, you need to configure the parameters again. If you press the y key, the following instruction will be displayed. Enter the calibration item.

```
===== Calibration Item ======
 1: Offset
 2: Half scale range
 3: Gain
============================

Enter the calibration item: 1
```

Mode	Calibration Order
PCI expansion boards (PCI series) PCI-3176, PCI-3310, PCI-3335, PCI-3336, PCI-3337, PCI-3340, PCI-3347 **PCI expansion boards (PAZ series)** PAZ-3176, PAZ-3310, PAZ-3336, PAZ-3340 **CompactPCI expansion boards** CTP-3340A, CTP-3340B, CTP-3340C, CTP-3340D, CTP-3347	1st: Offset 2nd: Gain 3rd: Half scale range
PCI expansion boards (PCI series) PCI-3174, PCI-3175, PCI-3305, PCI-3325, PCI-3329, PCI-3338, PCI-3341A, PCI-3342A, PCI-3343A, PCI-3345A, PCI-3346A, PCI-3521, PCI-3522A, PCI-3523A, PCI-3525 **PCI expansion boards (PAZ series)** PAZ-3174, PAZ-3305, PAZ-3325, PAZ-3329, PAZ-3338, PAZ-3521 **CompactPCI expansion boards** CTP-3174, CTP-3175, CTP-3182, CTP-3325, CTP-3329, CTP-3338, CTP-3342, CTP-3343, CTP-3346, CTP-3348, CTP-3349, CTP-3350, CTP-3351, CTP-3521, CTP-3522, CTP-3523	No applicable

4. The configration information will be displayed. Then connect the multimeter to the channel according to the instructions on the screen.

```
================================================
  Channel |      Range     | Calibration Item

        1  | -5 V to +5 V   | Offset

=== Connection to the Multimeter ===============

 Connect channel 1 to your multimeter.

 After the connection is completed,
 Pres the y key:

 Pres the y key after ready to run: y
```

5. Apply the voltage to the calibration channel with the accurate voltage supply. Press the y key after you ready to run.

7.1.5 Calibrating the On-Board Potentiometer

This program displays the goal and tolerance of voltage, and you can calibrate the potentiometer according to the instructions on the screen.

```
==== Target Value ====================
Max.   -4.998932 V
          |
Goal:  -4.999237 V
          |
Min.   -4.999542 V

==================================

Press the u key to increase the volume:
Press the d key to decrease the volume:
Press the n key to go next step: n
```

1. Press the u key to increase the volume.
   Press the d key to decrease the volume.
   Press the n key to go next step.

2. If you press the n key, the following message will appear. To save the settings, press the y key. If you don't need to save the settings, press the n key.

```
Save? (y/n): y
```

3. After you press the y or n key, the following instructions will appear. Press the n key to go next step. Press the q key to exit the program.

```
Press the n key to go next step:
Press the q key to exit: q
```

Chapter 8 Important Information

8.1 Limited Warranty

Interface does not warrant uninterrupted or error-free operations of the software product. The entire risks as to the quality of or arising out of use or performance of the software products, if any, remains with you.

Interface believes that information contained in the document is accurate. The document is carefully reviewed for technical accuracy. Interface reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. Interface is not liable for any damages arising out of or related to this document or the information contained in it.

Charts and tables contained in this document are only for illustration purposes and may vary depending upon a user's specific application.

All official specifications in metric. English unit supplied for convenience.

8.2 Copyrights and Intellectual Property Rights

Interface owns all titles and intellectual property rights in and to the products. The products includes computer software, may include audio/visual content such as images, text, or pictures.

No part of this publication may not be reproduced or altered in any form or by any means without written prior permission of Interface Corporation.

8.3 Warning Regarding Medical and Clinical Use of Interface Products

Interface products are not designed with components intended to ensure a level of reliability suitable for use under conditions that might cause serious injury or depth.

Interface products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of human.

Applications of Interface products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application engineer.

8.4 Limitation of Liability

Interface will not be liable for any special, incidental, indirect, or consequential damages whatsoever (including, but not limited to, damages for lost of profits or confidential or other information, for business interruption, for personal injury, for loss of privacy), even if interface, or any reseller has been advised of the possibility of such damages.

Customer's right to recover damages caused by fault or negligence on the part of Interface Corporation shall be limited to the amount paid by the customer for that product.

8.5 Trademark

Products and company names are trademarks, registered trademarks, or servicemarks of their respective owners.